Desconocido

## 12 Terminar tu primer juego: ¡Rebota!

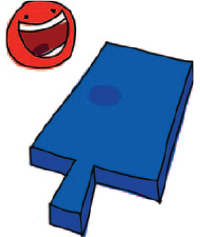


En el capítulo anterior, empezamos a construir nuestro primer juego, *¡Rebota* !, creando un lienzo y añadiendo una pelota que rebota a nuestro código. En este momento, nuestra pelota rebotará eternamente por la pantalla, lo que no tiene mucho sentido para un juego. En este capítulo, añadiremos una paleta para que la utilice el jugador. También añadiremos un elemento de azar al juego, que lo hará más desafiante y divertido.

### Añadir la paleta

Una pelota que rebota no tiene mucha gracia cuando no hay nada con lo que golpearla. Así que vamos a crear una paleta.

Empezaremos añadiendo el siguiente código directamente después de la clase Ball para crear una paleta (lo pondrás en una nueva línea debajo de la función draw de la pelota):



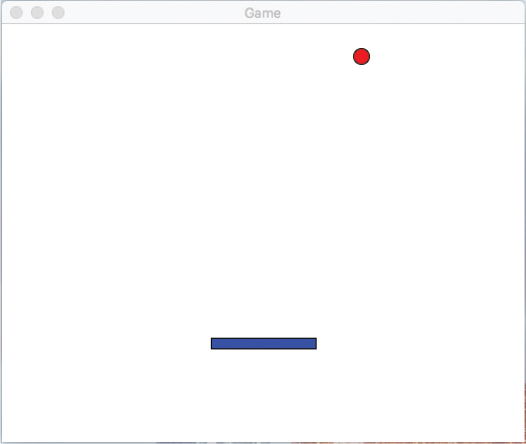
class Paddle:  
 def \_\_init\_\_(self, canvas, color):  
 self.canvas = canvas  
 self.id = canvas.create\_rectangle(0, 0, 100, 10, fill=color)  
 self.canvas.move(self.id, 200, 300)  
  
 def draw(self):  
 pass

Este código añadido es casi exactamente igual al que hicimos con nuestra primera versión de la clase Ball , salvo que llamamos a create\_rectangle (en lugar de a create\_oval ) y movemos el rectángulo a la posición 200, 300 (200 píxeles a lo ancho y 300 píxeles a lo bajo).

A continuación, al final de tu listado de código, crea un objeto de la clase Paddle , y luego cambia el bucle principal para llamar a la función draw de la paleta, como se muestra aquí:

➊ paddle = Paddle(canvas, 'blue')  
 ball = Ball(canvas, 'red')  
  
 while True:  
 ball.draw()  
 ➋ paddle.draw()  
 tk.update\_idletasks()  
 tk.update()  
 time.sleep(0.01)

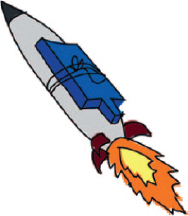
Los cambios se pueden ver en ➊ y ➋ . Si ejecutas ahora el juego, deberías ver la pelota rebotando y una paleta rectangular inmóvil ( [Figura 12-1](ch12.xhtml#ch12fig01) ).



Figura*12-1: P* elota y pala

### Cómo mover la paleta

Para hacer que la paleta se mueva a izquierda y derecha, utilizaremos eventos para vincular las teclas de flecha *izquierda* y *derecha* a nuevas funciones de la clase Paddle . Cuando el jugador pulse la tecla de flecha izquierda, la variable x se establecerá en -2 (para moverse a la izquierda). Al pulsar la tecla de flecha derecha, la variable x se establece en 2 (para moverse a la derecha).



El primer paso es añadir la variable de objeto x a la función \_\_init\_\_ de nuestra clase Paddle , así como una variable para la anchura del lienzo, como hicimos con la clase Ball :

def \_\_init\_\_(self, canvas, color):  
 self.canvas = canvas  
 self.id = canvas.create\_rectangle(0, 0, 100, 10, fill=color)   
 self.canvas.move(self.id, 200, 300)  
 ➊ self.x = 0  
 ➋ self.canvas\_width = self.canvas.winfo\_width()

Consulta ➊ y ➋ para ver los cambios. Ahora añadiremos las funciones para cambiar la dirección entre izquierda ( turn\_left ) y derecha ( turn\_right ) justo después de la función draw :

def turn\_left(self, evt):  
 self.x = -2  
  
 def turn\_right(self, evt):  
 self.x = 2

Podemos vincular estas funciones a la tecla correcta en la función \_\_init\_\_ de la clase con estas dos líneas. En "Hacer que un objeto reaccione a algo", en [la página 162](ch10.xhtml#ch10lev1sec13) , utilizamos la vinculación para hacer que Python llame a una función cuando se pulsa una tecla. En este caso, vinculamos la función turn\_left de nuestra clase Paddle a la tecla de la flecha izquierda, utilizando el nombre de evento <KeyPress-Left> . A continuación, vinculamos la función turn\_right a la tecla de flecha derecha, utilizando el nombre de evento <KeyPress-Right> . Nuestra función \_\_init\_\_ tiene ahora este aspecto:

def \_\_init\_\_(self, canvas, color):  
 self.canvas = canvas  
 self.id = canvas.create\_rectangle(0, 0, 100, 10, fill=color)  
 self.canvas.move(self.id, 200, 300)  
 self.x = 0  
 self.canvas\_width = self.canvas.winfo\_width()  
 ➊ self.canvas.bind\_all('<KeyPress-Left>', self.turn\_left)  
 ➋ self.canvas.bind\_all('<KeyPress-Right>', self.turn\_right)

Consulta ➊ y ➋ para ver los cambios. La función draw para la clase Paddle es similar a la de la clase Ball :

def draw(self):  
 self.canvas.move(self.id, self.x, 0)  
 pos = self.canvas.coords(self.id)  
 if pos[0] <= 0 or pos[2] >= self.canvas\_width:  
 self.x = 0

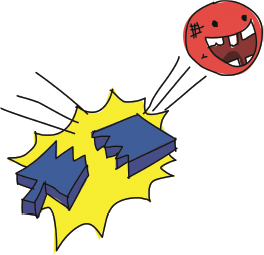
Utilizamos la función move de canvas para mover la paleta en la dirección de la variable x con self.canvas.move(self.id, self.x, 0) . A continuación, obtenemos las coordenadas de la paleta para ver si ha golpeado el lado izquierdo o derecho de la pantalla, utilizando el valor de pos . En lugar de rebotar como la pelota, la pala debe dejar de moverse. Así, cuando la *coordenada x* izquierda ( pos[0] ) es menor o igual que 0 ( <= 0 ), ponemos la variable x a 0 con self.x = 0 . Del mismo modo, cuando la *coordenada* x derecha ( pos[2] ) sea mayor o igual que la anchura del lienzo  ( >= self.canvas\_width ), también ponemos la variable x a 0.

Nota

*Si ejecutas el programa ahora, es posible que tengas que hacer clic en el lienzo antes de que el juego reconozca las acciones de las teclas de flecha izquierda y derecha.* Hacer clic en el lienzo le*da foco, lo que significa que sabe que debe hacerse cargo cuando alguien pulsa una tecla del teclado.*

### Averiguar cuándo la pelota golpea la pala

En este punto de nuestro código, la pelota no golpeará la pala. De hecho, la pelota volará directamente a través de la paleta. La pelota necesita saber cuándo ha golpeado la paleta, igual que necesita saber cuándo ha golpeado una pared.



Podríamos resolver este problema añadiendo código a la función draw (donde tenemos el código que comprueba si hay paredes), pero es mejor idea mover este código a nuevas funciones para dividir las cosas en trozos más pequeños. Si ponemos demasiado código en un solo lugar (dentro de una función, por ejemplo), podemos hacer que el código sea mucho más difícil de entender. Hagamos los cambios necesarios.

En primer lugar, cambiamos la función \_\_init\_\_ de la bola para que podamos pasar el objeto paddle como parámetro:

class Ball:  
 ➊ def \_\_init\_\_(self, canvas, paddle, color):  
 self.canvas = canvas  
 ➋ self.paddle = paddle  
 self.id = canvas.create\_oval(10, 10, 25, 25, fill=color)  
 self.canvas.move(self.id, 245, 100)  
 starts = [-3, -2, -1, 1, 2, 3]  
 self.x = random.choice(starts)  
 self.y = -3  
 self.canvas\_height = self.canvas.winfo\_height()   
 self.canvas\_width = self.canvas.winfo\_width()

Observa que cambiamos los parámetros de \_\_init\_\_ para incluir la paleta ➊ . A continuación, asignamos el parámetro de la paleta a la variable de objeto paddle ➋ .

Una vez guardado el objeto paddle , tenemos que cambiar el código donde creamos el objeto ball . Este cambio se encuentra al final de nuestro programa, justo antes del bucle principal while :

paddle = Paddle(canvas, 'blue')  
ball = Ball(canvas, paddle, 'red')  
  
while True:  
 ball.draw()  
 paddle.draw()  
 tk.update\_idletasks()  
 tk.update()  
 time.sleep(0.01)

Para ver si la pelota ha golpeado la paleta, necesitamos un código un poco más complicado que el código añadido anteriormente para comprobar si hay paredes. Llamaremos a esta función hit\_paddle y la invocaremos en la función draw de la clase Ball , donde comprobamos si la pelota ha tocado el fondo de la pantalla:

def draw(self):  
 self.canvas.move(self.id, self.x, self.y)  
 pos = self.canvas.coords(self.id)  
 if pos[1] <= 0 or pos[3] >= self.canvas\_height:  
 self.y = self.y \* -1  
 ➊ if self.hit\_paddle(pos) == True:  
 ➋ self.y = self.y \* -1  
 if pos[0] <= 0 or pos[2] >= self.canvas\_width:  
 self.x = self.x \* -1

En el código recién añadido, si hit\_paddle devuelve True ➊ , cambiamos la dirección de la bola ajustando la variable de objeto y a su valor multiplicado por -1 ➋ (lo mismo que cuando toca la parte superior o inferior de la tela). Con este código, básicamente estamos diciendo: "Si la pelota ( self ) ha golpeado la pala, invertimos su dirección vertical".

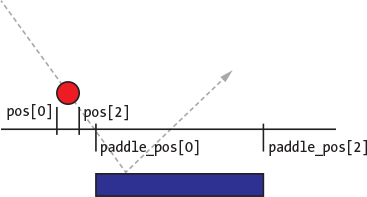
Podríamos combinar las comprobaciones de la parte superior, inferior y de la paleta en una sola declaración if -pero es más fácil para los nuevos programadores leer este código si las mantenemos separadas.

No intentes ejecutar el juego todavía; aún tenemos que crear la función hit\_paddle . Hagámoslo ahora. Añade la función hit\_paddle justo antes de la función draw en la clase Ball :

def hit\_paddle(self, pos):  
 paddle\_pos = self.canvas.coords(self.paddle.id)  
 if pos[2] >= paddle\_pos[0] and pos[0] <= paddle\_pos[2]:  
 if pos[3] >= paddle\_pos[1] and pos[3] <= paddle\_pos[3]:  
 return True  
 return False

En primer lugar, definimos la función con el parámetro pos . Este parámetro contiene las coordenadas actuales de la pelota. Después, obtenemos las coordenadas de la paleta y las almacenamos en la variable paddle\_pos .

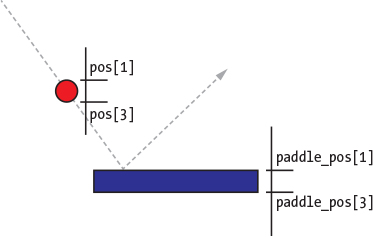
A continuación, tenemos la primera parte de nuestra sentencia if-then , que dice: "Si el lado derecho de la pelota es mayor que el lado izquierdo de la paleta, y el lado izquierdo de la pelota es menor que el lado derecho de la paleta. . ." Aquí, pos[2] contiene la *coordenada* x del lado derecho de la pelota, y pos[0] contiene la *coordenada* x de su lado izquierdo. La variable paddle\_pos[0] contiene la *coordenada* x del lado izquierdo de la paleta, y paddle\_pos[2] contiene la *coordenada* x del lado derecho. [La Figura 12-2](ch12.xhtml#ch12fig02) muestra el aspecto de estas coordenadas cuando la pelota está a punto de golpear la paleta.



*Figura 12-2: Pelota a punto de golpear la pala - mostrando coordenadas horizontales*

La pelota está cayendo hacia la paleta, pero en este caso, ves que el lado derecho de la pelota ( pos[2] ) aún no ha cruzado el lado izquierdo de la paleta (eso es paddle\_pos[0] ).

A continuación, vemos si la parte inferior de la bola ( pos[3] ) está entre la parte superior ( paddle\_pos[1] ) y la parte inferior ( paddle\_pos[3] ) de la pala. En la [Figura 12-3](ch12.xhtml#ch12fig03) , puedes ver que la parte inferior de la pelota ( pos[3] ) aún no ha golpeado la parte superior de la pala ( paddle\_pos[1] ).



Figura*12-3: Pelota a punto de golpear la pala - mostrando coordenadas verticales*

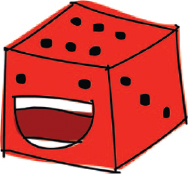
Así, basándose en la posición actual de la pelota, la función hit\_paddle devolvería False .

Nota

*¿Por qué necesitamos ver si la parte inferior de la pelota está entre la parte superior e inferior de la pala? ¿Por qué no ver simplemente si la parte inferior de la bola ha tocado la parte superior de la paleta? Porque cada vez que movemos la pelota por el lienzo, nos movemos a saltos de 3 píxeles. Si nos limitáramos a comprobar si la pelota ha llegado a la parte superior de la paleta ( pos[1] ), podríamos haber saltado más allá de esa posición. En ese caso, la bola seguiría viajando y atravesaría la paleta sin detenerse.*

### Añadir un elemento de azar

Ahora es el momento de convertir nuestro programa en un juego y no sólo en una pelota que rebota y una paleta. Los juegos necesitan un elemento *de* azar, o una forma de que el jugador pierda. En nuestro juego actual, la pelota rebotará eternamente, así que no hay nada que perder.



Terminaremos nuestro juego añadiendo código que diga que el juego termina si la pelota toca el fondo del lienzo (en otras palabras, cuando toque el suelo).

En primer lugar, añadimos la variable de objeto hit\_bottom al final de la función \_\_init\_\_ de la clase Ball :

self.canvas\_height = self.canvas.winfo\_height()  
 self.canvas\_width = self.canvas.winfo\_width()  
 self.hit\_bottom = False

A continuación, cambiamos el bucle principal en la parte inferior del programa, de la siguiente manera:

while True:  
 ➊ if ball.hit\_bottom == False:  
 ball.draw()  
 paddle.draw()  
 tk.update\_idletasks()  
 tk.update()  
 time.sleep(0.01)

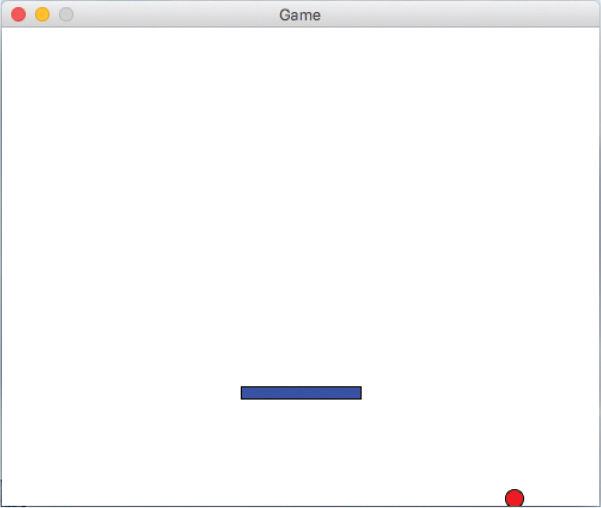
Ahora el bucle sigue comprobando hit\_bottom ➊ para ver si efectivamente la pelota ha tocado el fondo de la pantalla. El código debe seguir moviendo la pelota y la pala sólo si la pelota no ha tocado el fondo, como puedes ver en la declaración if anterior. El juego termina cuando la pelota y la paleta dejan de moverse. (Ya no los animamos.)

El último cambio es en la función draw de la clase Ball :

def draw(self):  
 self.canvas.move(self.id, self.x, self.y)  
 pos = self.canvas.coords(self.id)  
 if pos[1] <= 0:  
 self.y = self.y \* -1  
 ➊ if pos[3] >= self.canvas\_height:  
 self.hit\_bottom = True  
 if self.hit\_paddle(pos) == True:  
 self.y = self.y \* -1  
 if pos[0] <= 0 or pos[2] >= self.canvas\_width:  
 self.y = self.y \* -1

Modificamos la sentencia if para ver si la pelota ha tocado el fondo de la pantalla (es decir, si la posición de la pelota es mayor o igual que canvas\_height ) ➊ . Si es así, en la línea siguiente, establecemos hit\_bottom en True , en lugar de cambiar el valor de la variable y , porque no hay necesidad de hacer rebotar la pelota una vez que toca el fondo de la pantalla.

Cuando ejecutes el juego ahora y falles al golpear la pelota con la pala, todo el movimiento de la pantalla debería detenerse. El juego debería terminar una vez que la pelota toque el fondo del lienzo, como se muestra en la [Figura 12-4 .](ch12.xhtml#ch12fig04)



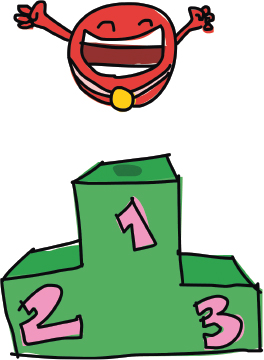
*Figura 12-4:* La pelota toca*el* fondo de la pantalla

Tu programa debería parecerse ahora al siguiente código. Si tienes problemas para que tu juego funcione, comprueba lo que has introducido con este código:

from tkinter import \*  
import random  
import time  
  
class Ball:  
 def \_\_init\_\_(self, canvas, paddle, color):  
 self.canvas = canvas  
 self.paddle = paddle  
 self.id = canvas.create\_oval(10, 10, 25, 25, fill=color)  
 self.canvas.move(self.id, 245, 100)  
 starts = [-3, -2, -1, 1, 2, 3]  
 self.x = random.choice(starts)  
 self.y = -3  
 self.canvas\_height = self.canvas.winfo\_height()  
 self.canvas\_width = self.canvas.winfo\_width()  
 self.hit\_bottom = False  
   
 def hit\_paddle(self, pos):  
 paddle\_pos = self.canvas.coords(self.paddle.id)  
 if pos[2] >= paddle\_pos[0] and pos[0] <= paddle\_pos[2]:  
 if pos[3] >= paddle\_pos[1] and pos[3] <= paddle\_pos[3]:  
 return True  
 return False  
   
 def draw(self):  
 self.canvas.move(self.id, self.x, self.y)  
 pos = self.canvas.coords(self.id)  
 if pos[1] <= 0:  
 self.y = self.y \* -1  
 if pos[3] >= self.canvas\_height:  
 self.hit\_bottom = True  
 if self.hit\_paddle(pos) == True:  
 self.y = self.y \* -1  
 if pos[0] <= 0 or pos[2] >= self.canvas\_width:  
 self.y = self.y \* -1  
  
class Paddle:  
 def \_\_init\_\_(self, canvas, color):  
 self.canvas = canvas  
 self.id = canvas.create\_rectangle(0, 0, 100, 10, fill=color)  
 self.canvas.move(self.id, 200, 300)  
 self.x = 0  
 self.canvas\_width = self.canvas.winfo\_width()  
 self.canvas.bind\_all('<KeyPress-Left>', self.turn\_left)  
 self.canvas.bind\_all('<KeyPress-Right>', self.turn\_right)  
  
 def draw(self):  
 self.canvas.move(self.id, self.x, 0)  
 pos = self.canvas.coords(self.id)  
 if pos[0] <= 0 or pos[2] >= self.canvas\_width:  
 self.x = 0  
   
 def turn\_left(self, evt):  
 self.x = -2  
  
 def turn\_right(self, evt):  
 self.x = 2  
  
tk = Tk()  
tk.title('Bounce Game')  
tk.resizable(0, 0)  
tk.wm\_attributes('-topmost', 1)  
canvas = Canvas(tk, width=500, height=400, bd=0, highlightthickness=0)  
canvas.pack()  
tk.update()  
  
paddle = Paddle(canvas, 'blue')  
ball = Ball(canvas, paddle, 'red')  
  
while True:  
 if ball.hit\_bottom == False:  
 ball.draw()  
 paddle.draw()  
 tk.update\_idletasks()  
 tk.update()  
 time.sleep(0.01)

### Lo que has aprendido

En este capítulo, hemos terminado de crear nuestro primer juego utilizando el módulo tkinter . Creamos clases para la paleta utilizada en nuestro juego, y utilizamos coordenadas para comprobar cuándo la pelota golpea la paleta o las paredes de nuestro lienzo de juego. Utilizamos eventos para vincular las teclas de flecha izquierda y derecha al movimiento de la paleta, y utilizamos un bucle principal para llamar a la función draw , para animarla. Por último, cambiamos nuestro código para añadir un elemento de azar, de modo que cuando el jugador falle la bola y ésta toque el fondo del lienzo, se acabe el juego.



### Programando puzzles

De momento, nuestro juego es bastante sencillo. Podrías cambiar muchas cosas para crear un juego más interesante. Intenta mejorar tu código de las siguientes maneras, y luego compara tus respuestas con las soluciones en http://python-for-kids.com [*.*](http://python-for-kids.com)

#### #1: Retrasar el inicio del juego

Nuestro juego se inicia rápidamente, y tienes que hacer clic en el lienzo antes de que reconozca la pulsación de las teclas de flecha izquierda y derecha del teclado. ¿Puedes añadir un retraso al inicio del juego para dar al jugador tiempo suficiente para hacer clic en el lienzo? O mejor aún, ¿puedes añadir un evento vinculante para el clic del ratón, que inicie el juego sólo entonces?

Pista 1: Ya has añadido eventos a la clase Paddle , así que ése puede ser un buen punto de partida.

Pista 2: El evento para el botón izquierdo del ratón es la cadena ’<Button-1>’ .

#### #2: Un "Game Over" adecuado

Todo se congela cuando el juego termina, lo que no es muy agradable para el jugador. Prueba a añadir el texto "Game Over" cuando la bola toque el fondo de la pantalla. Puedes utilizar la función create\_text , pero también puede resultarte útil el parámetro con nombre state (toma valores como normal y hidden ). Echa un vistazo a itemconfig en "Más formas de utilizar el identificador" en [la página 165 .](ch10.xhtml#ch10lev1sec14) Como reto adicional, añade un retardo para que el texto no aparezca inmediatamente.

#### #3: Acelera la pelota

En el tenis, cuando una pelota golpea tu raqueta, a veces sale volando más rápido que la velocidad a la que llegó, dependiendo de la fuerza con la que hagas el swing. En nuestro juego, la pelota va a la misma velocidad, se mueva o no la pala. Prueba a cambiar el programa para que la velocidad de la pala se transmita a la velocidad de la pelota.

#### #4: Registra la puntuación del jugador

¿Qué te parece registrar la puntuación? Cada vez que la pelota golpee la pala, la puntuación debería aumentar. Prueba a mostrar la puntuación en la esquina superior derecha del lienzo. Puedes consultar itemconfig en "Más formas de utilizar el identificador" en [la página 165](ch10.xhtml#ch10lev1sec14) para obtener una pista.