Desconocido

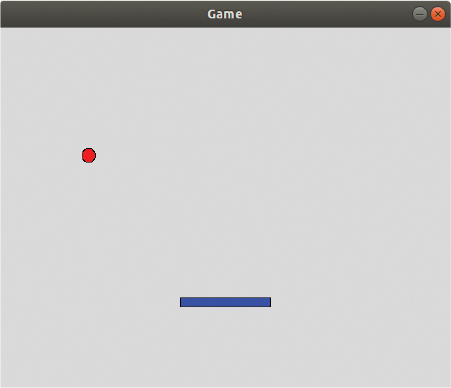
## 11 Comienza tu primer juego: ¡Rebota!



Hasta ahora, hemos tratado los fundamentos de la programación informática. Has aprendido a utilizar variables para almacenar información, sentencias if para código condicional y bucles for para repetir código. Sabes cómo crear funciones para reutilizar tu código, y cómo utilizar clases y objetos para dividir tu código en trozos más pequeños que sean más fáciles de entender. Has aprendido a dibujar gráficos en la pantalla con los módulos turtle y tkinter . Ahora es el momento de utilizar esos conocimientos para crear tu primer juego.

### Golpea la pelota que rebota

Vamos a desarrollar un juego con una pelota que rebota y una pala. La pelota volará por la pantalla y el jugador la hará rebotar en la paleta. Si la pelota toca el fondo de la pantalla, el juego termina. [La Figura 11](ch11.xhtml#ch11fig01) -1 muestra una vista previa del juego terminado.



*Figura 11-1: Juego ¡Rebota!*

Nuestro juego puede parecer bastante sencillo, pero el código será un poco más complicado que lo que hemos escrito hasta ahora, porque tiene que manejar muchas cosas. Por ejemplo, tiene que animar la paleta y la pelota, y detectar cuándo la pelota golpea la paleta o las paredes.

En este capítulo, empezaremos a crear el juego añadiendo un lienzo de juego y una pelota que rebota. En el próximo capítulo, completaremos el juego añadiendo la paleta.

### Crear el lienzo del juego

Para crear tu juego, abre primero un nuevo archivo en IDLE eligiendo **Archivo** **▸ Nuevo Archivo .** A continuación, importa tkinter y crea un canvas para dibujar en él:

from tkinter import \*  
 import random  
 import time  
 tk = Tk()  
➊ 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()

Este código es un poco diferente de los ejemplos anteriores. En primer lugar, importamos los módulos time y random con import random y import time , para utilizarlos un poco más adelante en el código. El módulo random proporciona (entre otras cosas) funciones para crear números aleatorios, y time tiene una función útil que le dirá a Python que detenga lo que está haciendo durante un periodo de tiempo.

Con tk.title(’Bounce Game’) ➊ , utilizamos la función title del objeto tk que creamos con tk = Tk() para dar un título a la ventana. Luego utilizamos resizable para que la ventana tenga un tamaño fijo. Los parámetros ( 0, 0 ) dicen: "El tamaño de la ventana no se puede cambiar ni horizontal ni verticalmente". A continuación, llamamos a los atributos wm\_ para decirle a tkinter que coloque la ventana que contiene nuestro lienzo delante de todas las demás ventanas ( ‘-topmost’ ).

Cuando creamos un objeto Canvas , le pasamos algunos parámetros con nombre más que en los ejemplos anteriores. Por ejemplo, tanto bd=0 como highlightthickness=0 se aseguran de que no haya ningún borde alrededor del exterior del lienzo, lo que hace que se vea mejor en la pantalla de nuestro juego. La línea canvas.pack() indica al lienzo que se ajuste a los parámetros de anchura y altura indicados en la línea anterior. Por último, tk.update() indica a tkinter que se inicialice para la animación de nuestro juego. Sin esta última línea, nada funcionaría como se espera.

Asegúrate de guardar tu código a medida que avanzas. Dale un nombre de archivo significativo la primera vez que lo guardes, como *paddleball.py .*



### Crear la clase Pelota

Ahora crearemos la clase de la pelota. Empezaremos con el código que necesitamos para que la pelota se dibuje a sí misma en el lienzo. Tenemos que hacer lo siguiente

1. Crear un class llamado Ball que tome como parámetros el lienzo y el color de la pelota que vamos a dibujar.
2. Guarda el lienzo como una variable objeto porque dibujaremos nuestra bola en él.
3. Dibuja un círculo relleno en el lienzo utilizando el valor del parámetro color como color de relleno.
4. Guarda el identificador que devuelve tkinter cuando dibuja el círculo (óvalo) porque vamos a utilizarlo para mover la bola por la pantalla.
5. Mueve el óvalo al centro del lienzo.

Este código debe añadirse justo después de las tres primeras líneas del archivo (después de import time ):

from tkinter import \*  
import random  
import time  
  
class Ball:  
 def \_\_init\_\_(self, canvas, color):  
 self.canvas = canvas  
 self.id = canvas.create\_oval(10, 10, 25, 25, fill=color)  
 self.canvas.move(self.id, 245, 100)  
  
 def draw(self):  
 pass

En primer lugar, nombramos a nuestra clase Ball . Luego creamos una función de inicialización (como se describe en "Inicializar un objeto" en [la página 109](ch08.xhtml#ch08lev1sec11) ) que toma los parámetros canvas y color . Establecemos la variable de objeto canvas con el valor del parámetro del mismo nombre. A continuación, llamamos a la función create\_oval con cinco parámetros: coordenadas *x e* *y* de la esquina superior izquierda (10 y 10), *coordenadas* *x e* *y* de la esquina inferior derecha (25 y 25) y el color de relleno del óvalo.

La función create\_oval devuelve un identificador para la forma que ha dibujado, que almacenamos en la variable de objeto id . Movemos el óvalo al centro del lienzo (coordenadas 245, 100). El lienzo sabe qué mover porque utilizamos el identificador de forma almacenado ( id ) para identificarlo.

En las dos últimas líneas de la clase Ball , creamos la función draw con def draw(self) , y el cuerpo de la función es simplemente la palabra clave pass . De momento, no hace nada, pero en breve añadiremos más cosas a esta función.



Ahora que hemos creado nuestra clase Ball , necesitamos crear un objeto de esta clase (recuerda: una clase describe lo que puede hacer, pero el objeto es lo que realmente lo hace). Añade este código al final del programa para crear un objeto bola roja:

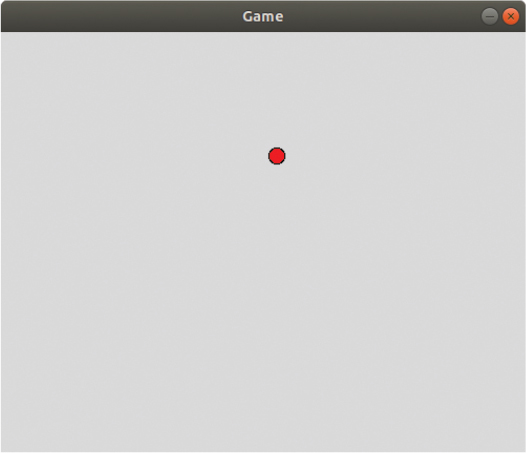
ball = Ball(canvas, 'red')

Puedes ejecutar este programa utilizando **Ejecutar** **▸ Ejecutar Módulo .** Si haces esto fuera de IDLE, el lienzo aparecerá durante una fracción de segundo y luego desaparecerá. Para evitar que la ventana se cierre inmediatamente, tenemos que añadir un bucle de animación, que se llama *bucle* principal de nuestro juego. (IDLE ya tiene un bucle principal, por eso la ventana no desaparece cuando lo ejecutas allí).

Un bucle principal es la parte central de un programa que generalmente controla la mayor parte de lo que hace. Nuestro bucle principal, de momento, sólo le dice a tkinter que redibuje la pantalla. El bucle, también llamado *bucle infinito* , sigue ejecutándose para siempre (o al menos hasta que cerremos la ventana), diciéndole constantemente a tkinter que redibuje la pantalla y durmiendo después durante una centésima de segundo mediante time.sleep . Añadiremos este código al final de nuestro programa:

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

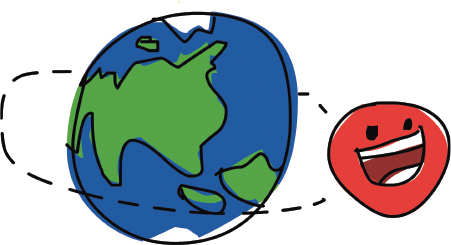
Ahora, si ejecutas el código, la bola debería aparecer en el centro del lienzo, como se muestra en la Figura 11-2 [.](ch11.xhtml#ch11fig02)



Figura*11-2:* Pelota en el centro del lienzo

### Añadir alguna acción

Ahora que tenemos configurada la clase Ball , es hora de animar la pelota. Haremos que se mueva, rebote y cambie de dirección.



#### Mover la pelota

Para mover la pelota, modifica la función draw como se indica a continuación:

class Ball:  
 def \_\_init\_\_(self, canvas, color):  
 self.canvas = canvas  
 self.id = canvas.create\_oval(10, 10, 25, 25, fill=color)  
 self.canvas.move(self.id, 245, 100)  
  
 def draw(self):  
 self.canvas.move(self.id, 0, -1)

Como \_\_init\_\_ guardó el parámetro canvas como variable del objeto canvas , utilizamos esa variable con self.canvas y llamamos a la función move en el lienzo.

Pasamos tres parámetros a move : el id del óvalo, y los números 0 y -1 . El 0 indica a la bola que no se mueva horizontalmente, y el -1 indica a la bola que se mueva 1 píxel hacia arriba en la pantalla.

Hacemos este pequeño cambio porque es beneficioso ir probando cosas sobre la marcha. Imagina que escribimos todo el código de nuestro juego de una vez y luego descubrimos que no funciona. ¿Por dónde empezaríamos a buscar para averiguar qué falló?

También cambiaremos el bucle principal al final de nuestro programa. En el bloque del bucle while (nuestro bucle principal), añadimos una llamada a la función draw del objeto ball , de esta forma:

while True:  
 ball.draw()  
 tk.update\_idletasks()  
 tk.update()  
 time.sleep(0.01)

Si ejecutas este código ahora, la pelota debería moverse por el lienzo y desaparecer de la parte superior de la pantalla: los comandos update \_idletasks y update indican a tkinter que se dé prisa en dibujar lo que hay en el lienzo.

El comando time.sleep es una llamada a la función sleep del módulo time , que le dice a Python que duerma durante una centésima de segundo ( 0.01 ). Esto garantiza que nuestro programa no se ejecute tan rápido que la bola desaparezca antes incluso de que la veas.

Este bucle está diciendo básicamente: "Mueve la bola un poco, vuelve a dibujar la pantalla con la nueva posición, duerme un momento y vuelve a empezar".

**NOTA**

*Puede que veas mensajes de error escritos en la Shell de Python cuando cierres la ventana del juego. Esto se debe a que al cerrar la ventana se interrumpe lo que tkinter está haciendo, y Python se está quejando de ello. Podemos ignorar con seguridad este tipo de errores.*

Ahora tu código de juego debería tener este aspecto:

from tkinter import \*  
import random  
import time  
  
class Ball:  
 def \_\_init\_\_(self, canvas, color):  
 self.canvas = canvas  
 self.id = canvas.create\_oval(10, 10, 25, 25, fill=color)  
 self.canvas.move(self.id, 245, 100)  
   
 def draw(self):  
 self.canvas.move(self.id, 0, -1)  
  
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()  
  
ball = Ball(canvas, 'red')  
  
while True:  
 ball.draw()  
 tk.update\_idletasks()  
 tk.update()  
 time.sleep(0.01)

Si ejecutas este código, la pelota empezará a moverse hacia arriba y saldrá por la parte superior de la ventana.

#### Hacer que la pelota rebote

Una pelota que desaparece de la parte superior de la pantalla no es especialmente útil para nuestro juego, así que hagamos que rebote. En primer lugar, guardaremos unas cuantas variables de objeto adicionales en la función de inicialización de la clase Ball , como se indica a continuación:

def \_\_init\_\_(self, canvas, color):  
 self.canvas = canvas  
 self.id = canvas.create\_oval(10, 10, 25, 25, fill=color)  
 self.canvas.move(self.id, 245, 100)  
 self.x = 0  
 self.y = -1  
 self.canvas\_height = self.canvas.winfo\_height()

Hemos añadido tres líneas más a nuestro programa. Con self.x = 0 , establecemos la variable de objeto x en 0; con self.y = -1 , establecemos la variable y en -1. Por último, fijamos la variable de objeto canvas\_height llamando a la función de lienzo winfo\_height . Esta función devuelve la altura actual del lienzo.

A continuación, volvemos a cambiar la función draw :

def draw(self):  
 self.canvas.move(self.id, self.x, self.y)  
 pos = self.canvas.coords(self.id)  
 if pos[1] <= 0:  
 self.y = 1   
 if pos[3] >= self.canvas\_height:  
 self.y = -1

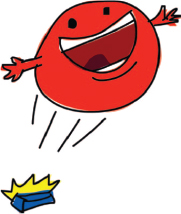
Primero cambiamos la llamada a la función move del lienzo pasando las variables de objeto x y y . A continuación, creamos una variable llamada pos llamando a la función coords del lienzo. Esta función devuelve las *coordenadas* *x e* *y* actuales de cualquier cosa dibujada en el lienzo, siempre que conozcas su número identificativo. En este caso, pasamos a coords la variable objeto id , que contiene el identificador del óvalo.

La función coords devuelve las coordenadas como una lista de cuatro números. Si imprimimos los resultados de llamar a esta función, veremos algo como esto

print(self.canvas.coords(self.id))  
[255.0, 29.0, 270.0, 44.0]

Los dos primeros números de la lista ( 255.0 y 29.0 ) contienen las coordenadas superiores izquierdas del óvalo ( *x1* e *y1* ), y el segundo par ( 270.0 y 44.0 ) son las coordenadas inferiores derechas *x2* e *y2* . Utilizaremos estos valores en las siguientes líneas de código.

Continuamos nuestro código viendo si la coordenada *y1* (¡esa es la parte superior de la bola!) es menor o igual que 0. Si es así, establecemos la variable de objeto y en 1 . En efecto, estamos diciendo que si golpeas la parte superior de la pantalla, tkinter dejará de restar 1 a la posición vertical, y la bola dejará de moverse hacia arriba (es una versión simple de la *detección de colisiones* ).

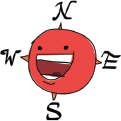


A continuación, vemos si la coordenada *y2* (¡esa es la parte inferior de la bola!) es mayor o igual que la variable canvas\_height . Si lo es, volvemos a poner la variable de objeto y en -1 . Ahora la bola dejará de moverse hacia abajo y volverá a subir.

Ejecuta ahora este código y la pelota debería rebotar arriba y abajo por el lienzo hasta que cierres la ventana.

#### Cambiar la dirección inicial de la bola

Hacer que una pelota rebote lentamente hacia arriba y hacia abajo no es un gran juego, así que mejoremos un poco las cosas cambiando la dirección inicial de la pelota, es decir, el ángulo en que se mueve cuando empieza el juego.



En la función \_\_init\_\_ , cambia estas dos líneas

self.x = 0  
 self.y = -1

por el código siguiente (asegúrate de que tienes el número correcto de espacios -hay ocho- al principio de cada línea):

starts = [-3, -2, -1, 1, 2, 3]  
 self.x = random.choice(starts)  
 self.y = 3

Comenzamos creando la variable starts con una lista de seis números. A continuación, establecemos el valor de la variable x , utilizando la función random.choice , que devuelve un elemento aleatorio de una lista. Al utilizar esa función, x puede ser cualquier número de la lista, de -3 a 3.

Por último, cambiamos y a -3 (para que la bola empiece el juego moviéndose hacia arriba). Ahora nuestra bola puede moverse en cualquier dirección, pero tenemos que hacer algunos añadidos más para asegurarnos de que no se desvanecerá por el lateral de la pantalla. Añade la siguiente línea al final de la función \_\_init\_\_ para guardar la anchura del lienzo en una nueva variable de objeto canvas\_width :

self.canvas\_width = self.canvas.winfo\_width()

Utilizaremos esta nueva variable de objeto en la función draw para ver si la bola ha golpeado el lado izquierdo o derecho del lienzo:

if pos[0] <= 0 or pos[2] >= self.canvas\_width:  
 self.x = self.x \* -1

Si la posición más a la izquierda de la bola es menor o igual que 0, o la posición más a la derecha de la bola es mayor o igual que la anchura del lienzo, hacemos este pequeño y extraño cálculo self.x = self.x \* -1 . La variable x se establece en el valor actual de x multiplicado por -1. Así, si el valor de x es 2, el nuevo valor será -2. Si el valor de x es -3, el nuevo valor será 3. Por tanto, cuando la pelota golpee un lado, rebotará en la dirección opuesta. Podemos hacer una comprobación similar para la parte superior e inferior del lienzo, utilizando la altura del lienzo y multiplicando la variable y por -1. Tu función draw debería tener ahora este aspecto:

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

El programa completo debería tener este aspecto:

from tkinter import \*  
import random  
import time  
  
class Ball:  
 def \_\_init\_\_(self, canvas, color):  
 self.canvas = canvas  
 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()  
   
 def draw(self):  
 self.canvas.move(self.id, self.x, self.y)  
 pos = self.canvas.coords(self.id)  
 if pos[0] <= 0 or pos[2] >= self.canvas\_width:  
 self.x = self.x \* -1  
 if pos[1] <= 0 or pos[3] >= self.canvas\_height:  
 self.y = self.y \* -1  
  
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()  
  
ball = Ball(canvas, 'red')  
  
while True:  
 ball.draw()  
 tk.update\_idletasks()  
 tk.update()  
 time.sleep(0.01)

Guarda y ejecuta el código, y la pelota debería rebotar por la pantalla sin desaparecer.

### Lo que has aprendido

En este capítulo, hemos empezado a crear nuestro primer juego utilizando el módulo tkinter . Creamos un objeto Ball y lo animamos para que se moviera por la pantalla. Usamos coordenadas para comprobar cuándo la pelota golpea los lados del lienzo y hacer que rebote. También utilizamos la función choice del módulo random para que nuestra bola no empiece a moverse siempre en la misma dirección. En el próximo capítulo, completaremos el juego añadiendo la paleta.

### Rompecabezas de programación

#### #1: Cambiar colores

Prueba a cambiar el color inicial de la bola y el color de fondo del lienzo: prueba con distintas combinaciones de colores y mira cuáles te gustan.

#### #2: Colores intermitentes

Como hay un bucle en la parte inferior de nuestro código, debería ser bastante fácil cambiar el color de la bola mientras se mueve por la pantalla. Podemos añadir algún código al bucle que elija diferentes colores (piensa en la función choice que utilizamos anteriormente en el capítulo), y luego actualice el color de la bola (quizás llamando a una nueva función de nuestra clase Ball ). Para ello, tendrás que utilizar la función itemconfig en el lienzo (consulta "Más formas de utilizar el identificador" en [la página 165](ch10.xhtml#ch10lev1sec14) ).

#### #3: ¡Toma posiciones!

Intenta cambiar el código para que la bola empiece en una posición diferente en la pantalla. Podrías hacer que la posición fuera aleatoria utilizando el módulo random (consulta la sección "Dibujar un montón de rectángulos" en la [página](ch10.xhtml#ch10lev1sec6) 145 para ver un ejemplo de cómo utilizar la función randrange de ese módulo). Pero tendrás que asegurarte de que la bola no empiece demasiado cerca, o por debajo, de la paleta, lo que hará que el juego sea imposible de jugar.

#### #4: Añadir la paleta... ?

Basándote en el código que hemos creado hasta ahora, ¿puedes averiguar cómo añadir la paleta antes de llegar al siguiente capítulo? Si echas un vistazo al [Capítulo 10](ch10.xhtml#ch10) , tal vez puedas averiguar cómo dibujarla antes de seguir adelante. ¡Luego comprueba en las páginas siguientes si lo has hecho bien!