

Un clásico: PONG

Un poco de Historia

Como puede leerse en la Wikipedia, aunque existieron con anterioridad otros videojuegos no eran más que proyectos experimentales: **Pong** es considerado por muchos el primer videojuego moderno y antecesor de las arcade y consolas actuales. Ello es debido a que fue el primero en comercializarse a nivel masivo y a que no se ejecutaba en máquinas únicas.



Pong es un juego en dos dimensiones que simula un tenis de mesa. El jugador controla en el juego una paleta moviéndola verticalmente en la parte izquierda de la pantalla, y puede competir tanto contra un oponente controlado por el ordenador, como con otro jugador humano que controla una segunda paleta en la parte opuesta. Los jugadores pueden usar las paletas para pegarle a la pelota hacia un lado u otro. El objetivo consiste en que uno de los jugadores consiga más puntos que el oponente al finalizar el juego. Estos puntos se obtienen cuando el jugador adversario falla al devolver la pelota.

Nosotros vamos a desarrollar la versión de dos jugadores, dejando para más adelante la implementación del juego contra el propio ordenador.

¡Empecemos a escribir código!

pong01.py

El primer paso es partir del esqueleto de toda aplicación pygame con las características propias del juego que estamos abordando. Podría quedar algo así:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

# pong01.py
# Este programa es sólo el comienzo de un ping-pong muy simple

import pygame, sys
from pygame.locals import *

pygame.init()

BLANCO = (255,255,255)

visor = pygame.display.set_mode((800,600),FULLSCREEN)

while True:

    for evento in pygame.event.get():

        if evento.type == KEYDOWN and evento.key == K_ESCAPE:
            pygame.quit()
            sys.exit()

    visor.fill((0,0,0))

    pygame.draw.circle(visor, BLANCO, (50,50),4,0)

    pygame.display.update()
```

Empezamos importando las librerías e inicializando el entorno, como siempre, para pasar a definir el color de la pelota y de las raquetas.

```
BLANCO = (255,255,255)
```

A continuación, almacenamos en la variable **visor** la **surface** donde mostraremos el juego. Observa que es a pantalla completa y que la resolución que hemos decidido es de 800x600 píxeles.

```
visor = pygame.display.set_mode((800,600),FULLSCREEN)
```

Pasamos al bucle del juego.

```
while True:  
    ...
```

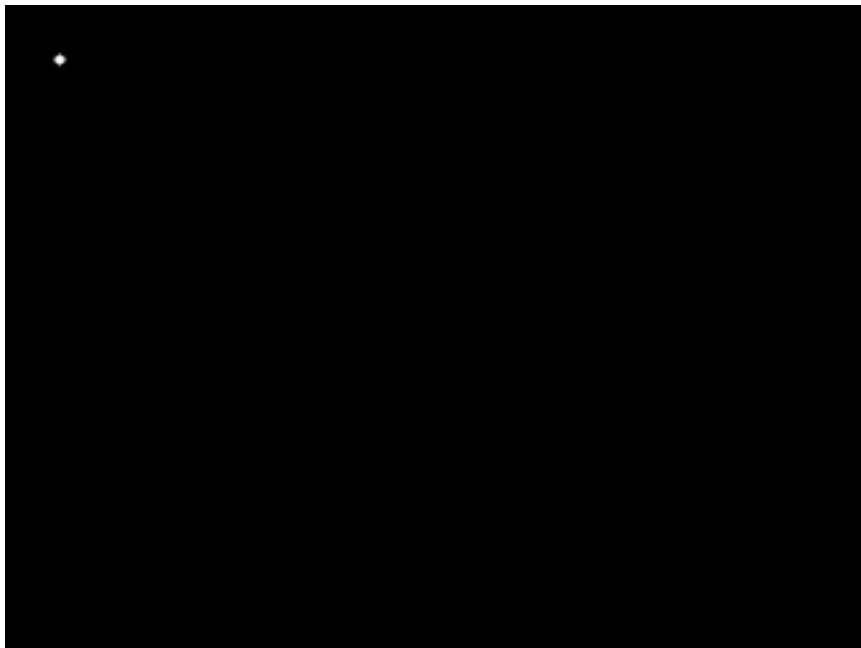
En primer lugar, dentro del bucle, tenemos a su vez el **bucle de eventos**. En él nos limitamos a ver si el usuario ha pulsado la tecla **ESC**, en cuyo caso el programa se termina.

```
for evento in pygame.event.get():  
  
    if evento.type == KEYDOWN and evento.key == K_ESCAPE:  
        pygame.quit()  
        sys.exit()
```

En segundo lugar, fuera del bucle de eventos y para finalizar, pasamos a dibujar el **frame**. Lo que hagamos aquí se hará en cada **fotograma** de la animación del juego, una y otra vez. De momento, sólo queremos que la pantalla esté en negro y que se dibuje en una posición fija la pelota; un círculo blanco, de 4 píxeles de radio y en la posición (50,50).

```
visor.fill((0,0,0))  
pygame.draw.circle(visor, BLANCO, (50,50),4,0)  
  
pygame.display.update()
```

Por supuesto, la última línea vuelca el frame entero en la pantalla de juego (o lo que es lo mismo, actualiza la surface de pygame). Éste, por el momento, será el resultado al ejecutar el programa:



pong02.py

Vamos a modificar ahora el programa para que la pelota se mueva. Los cambios, para que sean más evidentes, los pondremos en negrita:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

# pong02.py

# Modificación con la que la pelota se mueve.

import pygame, sys
from pygame.locals import *

pygame.init()

BLANCO = (255,255,255)
pelotaX = 50
pelotaY = 50

visor = pygame.display.set_mode((800,600),FULLSCREEN)

while True:

    for evento in pygame.event.get():

        if evento.type == KEYDOWN and evento.key == K_ESCAPE:
            pygame.quit()
            sys.exit()

        pelotaX += 5
        pelotaY += 5

    visor.fill((0,0,0))

    pygame.draw.circle(visor, BLANCO, (pelotaX,pelotaY),4,0)

    pygame.display.update()
```

La posición de la pelota ahora no es fija, así que necesitamos almacenar dónde está en dos variables, una para cada coordenada. Primero, claro, definimos sus valores iniciales (o lo que es lo mismo, la posición inicial de la pelota)

```
pelotaX = 50
pelotaY = 50
```

Para que la pelota se mueva, en cada frame tenemos que modificar su posición, es decir, los valores de sus coordenadas. Así que, dentro del bucle del juego (y fuera del bucle de eventos), incluimos

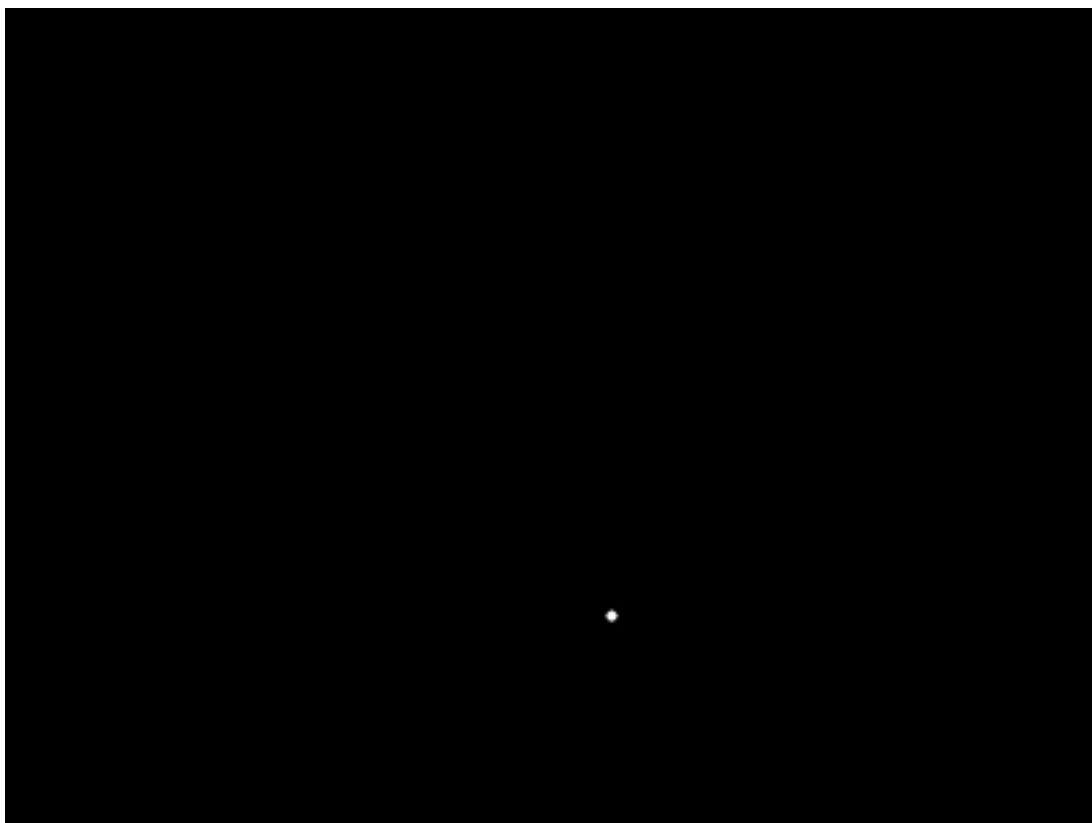
```
pelotaX += 5  
pelotaY += 5
```

Recuerda que **pelotaX += 5** es equivalente a **pelotaX = pelotaX + 5**. Al aumentar la variable de la posición X de la pelota en 5 hacemos que se desplace 5 píxeles a la derecha. Y al aumentar la posición Y en 5, conseguimos que se desplace 5 píxeles hacia abajo.

Por supuesto, el dibujo de la pelota ahora hay que hacerlo en su posición variable:

```
pygame.draw.circle(visor, BLANCO, (pelotaX,pelotaY),4,0)
```

Al ejecutar, ahora, el programa, veremos cómo la pelota se desplaza por la pantalla.



pong03.py

Tenemos un inconveniente grave, y es que si tu ordenador es rápido la pelota se mueve a tal velocidad que apenas puede verse. Este es un punto importante para el programador de videojuegos; siempre debe controlarse la velocidad de la animación para que sea jugable o realista. En la práctica esto se traduce en fijar el número de fotogramas por segundo que muestra el ordenador. Veamos cómo podemos implementarlo:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

# pong03.py
# Limitando la velocidad de la animación

import pygame, sys
from pygame.locals import *

pygame.init()

fps = 60
tiempo = 0
BLANCO = (255,255,255)
pelotaX = 50
pelotaY = 50

visor = pygame.display.set_mode((800,600),FULLSCREEN)

while True:

    if pygame.time.get_ticks()-tiempo < 1000/fps:
        continue
    tiempo = pygame.time.get_ticks()

    for evento in pygame.event.get():

        if evento.type == KEYDOWN and evento.key == K_ESCAPE:
            pygame.quit()
            sys.exit()

    pelotaX += 5
    pelotaY += 5

    visor.fill((0,0,0))

    pygame.draw.circle(visor, BLANCO, (pelotaX,pelotaY),4,0)

    pygame.display.update()
```

La idea es la siguiente; tenemos que mirar, en cada pasada del bucle del juego, si ha pasado el tiempo suficiente como para que toque dibujar. En caso afirmativo, se dibuja. En caso negativo, se salta el proceso y se pasa a la siguiente iteración del bucle. Para ello tenemos una instrucción estupenda en Python, **continue**, que dentro de un bucle **while** o **for** simplemente hace que se ignore el resto de las instrucciones y se pase al siguiente.

Pero, por supuesto, necesitamos unas variables que se encarguen de calcular y almacenar el tiempo transcurrido. Por ello, al principio del programa, hemos añadido

```
fps = 60
tiempo = 0
```

fps representa los **frames por segundo**, es decir, el número de fotogramas por segundo que queremos. La variable **tiempo** nos va a servir para tomar un tiempo de referencia. ¿Cómo es esto? Pygame tiene una función que nos devuelve el número de **ticks** (esto es, el número de milésimas de segundo) que han transcurrido desde que el sistema se inicializó con **pygame.init()**. Dicha función es **pygame.time.get_ticks()**. Para ver si ha pasado el tiempo suficiente desde uno de referencia dado, tendremos que hacer la resta entre ambos.

¿Cuánto tiempo tiene que pasar? Eso nos lo marca **fps**. Si queremos 10 fotogramas por segundo, deberá dibujarse un fotograma cada 1/10 segundos, es decir, cada 1000/10 milisegundos (en este caso, 100). Si queremos 20 fotogramas por segundo, deberá dibujarse en pantalla cada 1000/20 = 50 milisegundos. Y así sucesivamente.

¿Qué es lo que hay que hacer entonces? Mirar cuanto tiempo ha pasado y si no son los suficientes milisegundos (**ticks**), pasar. Observa cómo lo hemos implementado dentro del bucle while:

```
if pygame.time.get_ticks()-tiempo < 1000/fps:
    continue
tiempo = pygame.time.get_ticks()
```

En el **if** miramos si no ha pasado el tiempo suficiente respecto al tiempo de referencia (que, inicialmente, es 0, el comienzo). En tal caso, usamos la sentencia **continue** para que se ignore el resto del bucle y se proceda a una nueva pasada.

En el caso de que el **if** falle (es decir, **sí** ha pasado el tiempo necesario y **sí** debe dibujarse el fotograma), el bucle continúa y se ejecuta la siguiente instrucción. Ahora es importante tomar un nuevo tiempo de referencia ya que hay que comparar con éste preciso momento para dibujar el próximo fotograma. Y para ello almacenamos en la variable **tiempo** un nuevo valor que se corresponde con el actual.

Si ejecutas el programa, verás que la velocidad de la pelota ya es distinta y más razonable. Puedes variar a tu gusto el valor de fps para conseguir la velocidad óptima que desees (también puedes variar el valor que se suma a pelotaX y pelotaY).

En las diferentes versiones del programa que vamos a trabajar incluiremos siempre este código. Pero si tienes curiosidad, basta con que elimines estas líneas (o las comentes con una #) para ver cómo sería el juego sin limitar su velocidad...

pong04.py

Nuestra pelota se mueve, pero desaparece en cuanto llega al borde de la pantalla. Eso hay que solucionarlo y lo hacemos en la tercera versión de nuestro programa:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# pong04.py
# Consiguiendo el rebote en las paredes

import pygame, sys
from pygame.locals import *

pygame.init()

fps = 60
tiempo = 0
BLANCO = (255,255,255)
pelotaX = 50
pelotaY = 50
pelotaDX = 5
pelotaDY = 5

visor = pygame.display.set_mode((800,600),FULLSCREEN)

while True:
    if pygame.time.get_ticks()-tiempo < 1000/fps:
        continue
    tiempo = pygame.time.get_ticks()
    for evento in pygame.event.get():
        if evento.type == KEYDOWN and evento.key == K_ESCAPE:
            pygame.quit()
            sys.exit()

    if pelotaX < 5 or pelotaX > 795:
        pelotaDX = -pelotaDX
    if pelotaY < 5 or pelotaY > 595:
        pelotaDY = -pelotaDY
    pelotaX += pelotaDX
    pelotaY += pelotaDY

    visor.fill((0,0,0))

    pygame.draw.circle(visor, BLANCO, (pelotaX,pelotaY),4,0)

    pygame.display.update()
```

Bien. El fundamento está en invertir la dirección. Cuando, por ejemplo, la pelota llega al borde superior de la pantalla el movimiento hacia arriba debe convertirse en movimiento

hacia abajo. Y viceversa. De la misma manera, en los extremos laterales, hay que cambiar la dirección hacia la derecha con la dirección hacia la izquierda.

¿Cómo hacerlo? Para empezar, hay que diferenciar el movimiento vertical y el horizontal. Así que el incremento en esas direcciones en la posición de la pelota lo almacenamos en dos variables. Eso lo hacemos de la siguiente manera.

```
pelotaDX = 5  
pelotaDY = 5
```

La variable **pelotaDX** almacena el movimiento en dirección horizontal y **pelotaDY** en vertical. Por supuesto, y más adelante, la posición de la pelota se modificará entonces así:

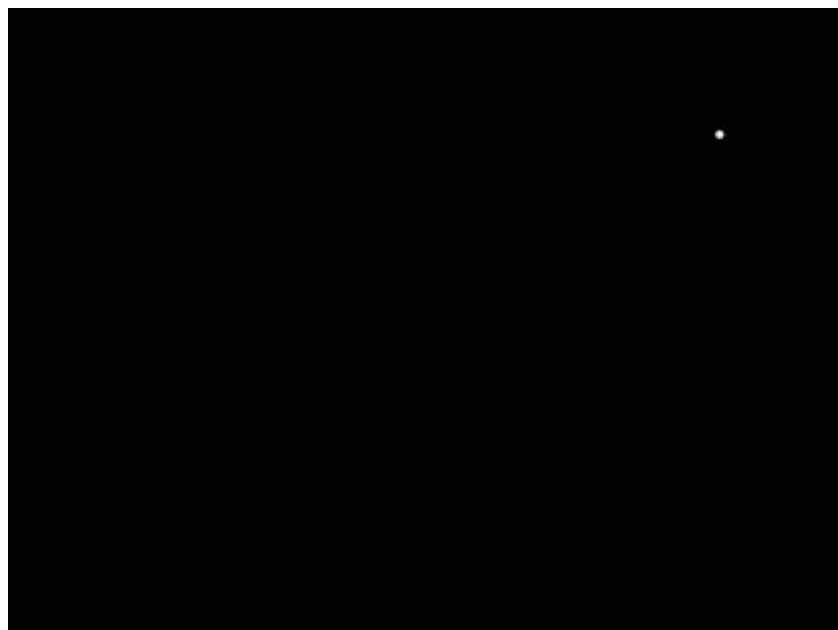
```
pelotaX += pelotaDX  
pelotaY += pelotaDY
```

Fijémonos de paso que si quisiéramos que la pelota se moviera más deprisa o más despacio, bastaría con poner diferentes valores en ambas variables.

Lo que queda ahora es invertir ese movimiento en los bordes de la pantalla. Eso requiere detectar cuándo la pelota está allí y cambiar el signo del movimiento (para que pase de sumar valores a las coordenadas a restar o viceversa). Es más fácil de ver que de describir:

```
if pelotaX < 5 or pelotaX > 795:  
    pelotaDX = -pelotaDX  
if pelotaY < 5 or pelotaY > 595:  
    pelotaDY = -pelotaDY
```

El primer if detecta si la pelota está en el borde izquierdo (su coordenada x es menor que 5) o en el borde derecho (es mayor que 795, recuerda que la resolución de pantalla que hemos elegido es de 800x600). En tal caso, se cambia el signo del movimiento horizontal de la pelota. El segundo if hace lo propio con los bordes superior e inferior.



pong05.py

Es el turno de que aparezca nuestra raqueta. La dibujaremos a la izquierda:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# pong05.py
# Aparece la raqueta

import pygame, sys
from pygame.locals import *

pygame.init()

fps = 60
tiempo = 0
BLANCO = (255,255,255)
pelotaX = 50
pelotaY = 50
pelotaDX = 5
pelotaDY = 5
raquetaX = 50
raquetaY = 250

visor = pygame.display.set_mode((800,600),FULLSCREEN)

while True:
    if pygame.time.get_ticks()-tiempo < 1000/fps:
        continue
    tiempo = pygame.time.get_ticks()
    for evento in pygame.event.get():
        if evento.type == KEYDOWN and evento.key == K_ESCAPE:
            pygame.quit()
            sys.exit()

    if pelotaX < 5 or pelotaX > 795:
        pelotaDX = -pelotaDX
    if pelotaY < 5 or pelotaY > 595:
        pelotaDY = -pelotaDY
    pelotaX += pelotaDX
    pelotaY += pelotaDY

    visor.fill((0,0,0))

    pygame.draw.circle(visor, BLANCO, (pelotaX,pelotaY),4,0)
    pygame.draw.rect(visor, BLANCO, (raquetaX,raquetaY,10,50))

    pygame.display.update()
```

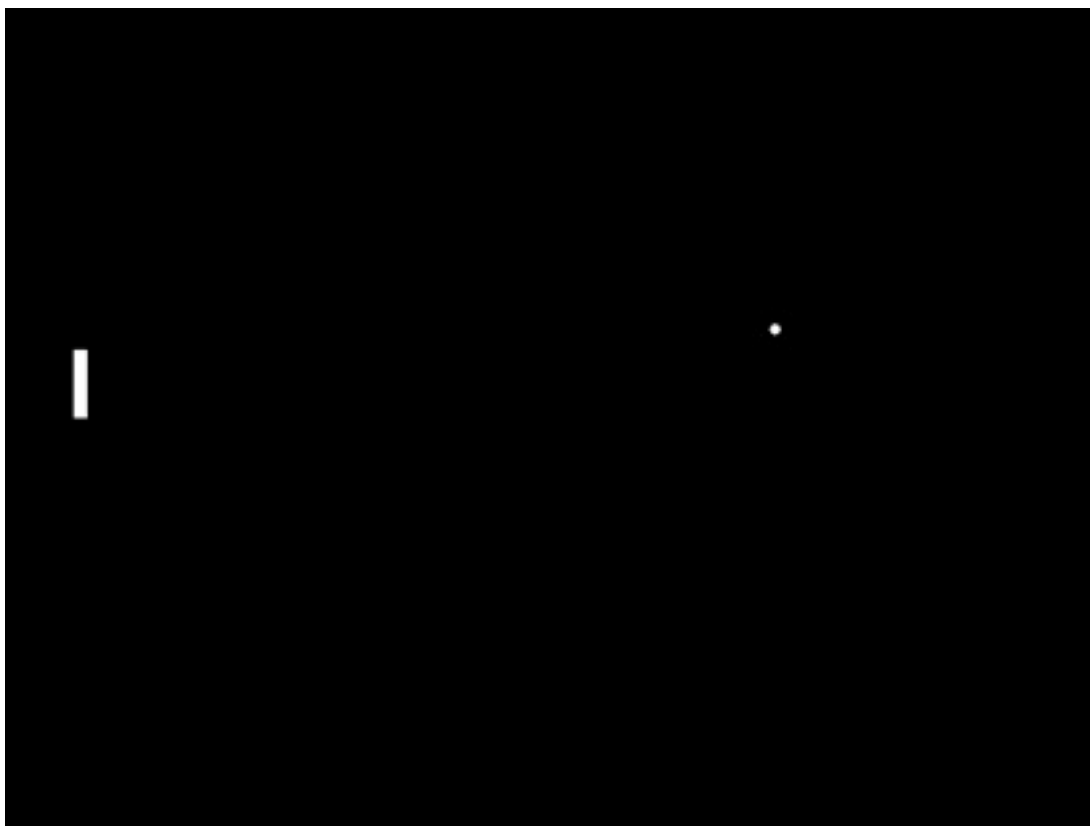
Como puede verse, y teniendo en cuenta que luego moveremos la raqueta, utilizamos las variables `raquetaX` y `raquetaY` para almacenar su posición:

```
raquetaX = 50  
raquetaY = 250
```

Y, más adelante, junto al dibujo de la pelota pasamos a dibujar la raqueta, un rectángulo blanco de 10 píxeles de ancho y 50 de largo:

```
pygame.draw.rect(visor, BLANCO, (raquetaX,raquetaY,10,50))
```

Éste es el resultado:



pong06.py

¡La raqueta la debemos poder mover! La quinta versión de nuestro juego se encarga:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# pong06.py
# La raqueta se mueve

import pygame, sys
from pygame.locals import *
pygame.init()

fps = 60
tiempo = 0
BLANCO = (255,255,255)
pelotaX = 50
pelotaY = 50
pelotaDX = 5
pelotaDY = 5
raquetaX = 50
raquetaY = 250
raquetaDY = 5

visor = pygame.display.set_mode((800,600),FULLSCREEN)
while True:
    if pygame.time.get_ticks()-tiempo < 1000/fps:
        continue
    tiempo = pygame.time.get_ticks()
    for evento in pygame.event.get():
        if evento.type == KEYDOWN:
            if evento.key == K_ESCAPE:
                pygame.quit()
                sys.exit()
            elif evento.key == K_a:
                raquetaY = raquetaY + raquetaDY
            elif evento.key == K_q:
                raquetaY = raquetaY - raquetaDY

    if pelotaX < 5 or pelotaX > 795:
        pelotaDX = -pelotaDX
    if pelotaY < 5 or pelotaY > 595:
        pelotaDY = -pelotaDY
    pelotaX += pelotaDX
    pelotaY += pelotaDY

    visor.fill((0,0,0))
    pygame.draw.circle(visor, BLANCO, (pelotaX,pelotaY),4,0)
    pygame.draw.rect(visor, BLANCO, (raquetaX,raquetaY,10,50))

    pygame.display.update()
```

Toca el turno de modificar el bucle de eventos. Para mover la raqueta tenemos que ver si se pulsa el teclado. Como nuestro juego va a ser de dos jugadores, uno jugará a la izquierda y el otro a la derecha. Empecemos por la izquierda y usaremos la **tecla q** para hacer que la raqueta suba y la **tecla a** para que baje. El movimiento de la raqueta va a ser sólo en vertical, así que usaremos una sola variable para indicarlo, almacenando en ella cuánto ha de moverse la raqueta cuando se pulse la tecla correspondiente:

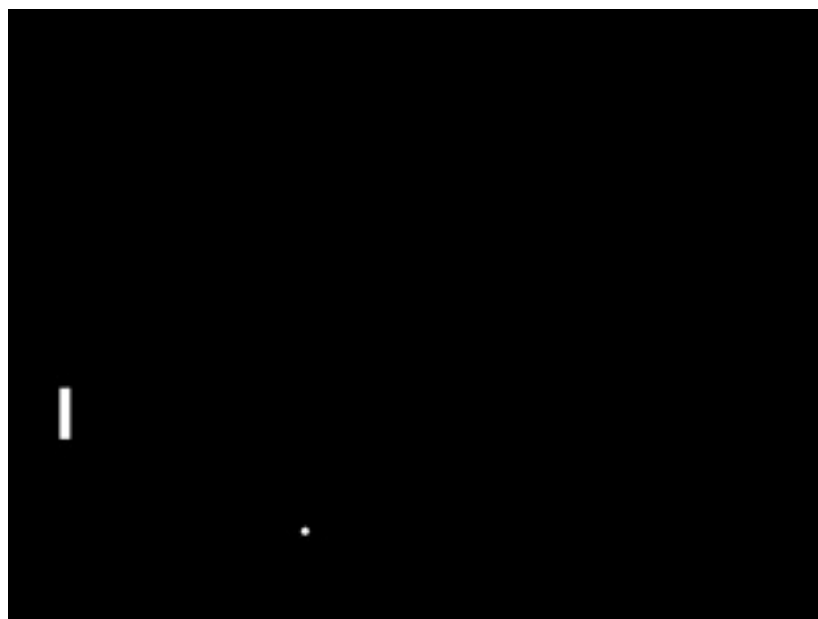
```
raquetaDY = 5
```

Bien. Sólo queda trabajar, como hemos dicho, en el bucle de eventos. En el apartado donde se mira qué teclas se han pulsado (recuerda que se mira si se ha pulsado la **tecla ESC** para salir del programa) hemos de añadir las del movimiento de la raqueta:

```
if evento.type == KEYDOWN:
    if evento.key == K_ESCAPE:
        pygame.quit()
        sys.exit()
    elif evento.key == K_a:
        raquetaY = raquetaY + raquetaDY
    elif evento.key == K_q:
        raquetaY = raquetaY - raquetaDY
```

El tipo de evento es el de pulsar una tecla (**KEYDOWN**) y si no es la **tecla ESC** la que se ha pulsado, hay que mirar si es la **tecla a** o la **tecla q**, de ahí que se use el **elif**. La sintaxis es muy sencilla, como puedes ver. La tecla pulsada Pygame la almacena en la variable **evento.key**, y para distinguir cual es, Pygame usa el convenio de hacer corresponder el nombre de la tecla con una constante que empieza por **K_** y añade la tecla. Así el nombre que da Pygame a la **tecla a** es **K_a**, a la **tecla q**, **K_q**, a la **tecla w**, **K_w**, etc.

En el caso de que la tecla sea la correcta, se sube o se baja la posición de la raqueta.



pong07.py

Cuando hayas ejecutado el juego tal como está en el apartado anterior, habrás notado un grave inconveniente; para mover la raqueta hay que pulsar cada vez la tecla y si la mantenemos pulsada, la raqueta no se mueve. Eso lo convierte en un 'juego injugable'.

El problema está en el bucle de eventos, pues hasta que no levantemos el dedo de la tecla y la volvamos a pulsar, no vuelve a producirse el evento deseado. Y eso no es lo que queremos... Lo que queremos es que mientras esté la tecla pulsada, la raqueta se siga moviendo en la dirección deseada.

La solución se puede abordar desde diferentes puntos de vista. Una de las más simples sería no cambiar directamente la posición de la raqueta en el bucle de eventos si no hacerlo fuera y dejar en ese bucle únicamente la modificación del valor que hay que sumarle o restarle luego a la hora de dibujar. Sería cambiar a algo así (ponemos sólo parte del código por que ésta no va a ser nuestra solución definitiva):

```
for evento in pygame.event.get():

    if evento.type == KEYDOWN:
        ...

        elif evento.key == K_a:
            raquetaDY = 5
        elif evento.key == K_q:
            raquetaDY = -5
    if evento.type == KEYUP:
        if evento.key == K_a or evento.key == K_q:
            raquetaDY = 0

    ...

    raquetaY += raquetaDY
```

Fíjate. Si se pulsa la **tecla a** el incremento va a ser de 5 pixeles hacia abajo, si es la **tecla q**, 5 pixeles hacia arriba. Por otra parte, si se produce el nuevo tipo de evento **KEYUP** (es decir, si se levanta el dedo de una tecla), el incremento se pone a 0. Observa, de paso, que nos aseguramos de que la tecla que se ha levantado es la correcta (ya que el juego no debe responder a otras teclas que no nos importan). Fuera del bucle de eventos, la posición de la raqueta se modifica según lo que indica la variable **raquetaDY**. ¡Eso es lo deseado! Si lo último que se ha hecho ha sido pulsar la **tecla q** o la **tecla a**, la variable tendrá el valor 5 o -5. Y así se mantendrá hasta que no se levante el dedo de la tecla, en cuyo caso el valor será 0. Todo debería funcionar bien...

Pero hay que anticiparse a los problemas que nos van a ir surgiendo con el desarrollo del juego. ¿Qué pasará cuando añadamos otro jugador? Habrá que incluir otra cantidad de código equivalente con las teclas correspondientes (digamos, la **tecla p** y la **tecla l**). Y hay algo más grave; ¿qué sucederá cuando los dos jugadores pulsen teclas a la vez? ¿A quién hará caso el bucle de eventos?

Nuestro abordaje definitivo va a ser diferente. Éste va a ser el código:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# pong07.py
# La raqueta se mueve correctamente

import pygame, sys
from pygame.locals import *

pygame.init()

fps = 60
tiempo = 0
BLANCO = (255,255,255)
pelotaX = 50
pelotaY = 50
pelotaDX = 5
pelotaDY = 5
raquetaX = 50
raquetaY = 250
raquetaDY = 5

visor = pygame.display.set_mode((800,600),FULLSCREEN)
while True:
    if pygame.time.get_ticks()-tiempo < 1000/fps:
        continue
    tiempo = pygame.time.get_ticks()
    for evento in pygame.event.get():
        if evento.type == KEYDOWN:
            if evento.key == K_ESCAPE:
                pygame.quit()
                sys.exit()

    if pelotaX < 5 or pelotaX > 795:
        pelotaDX = -pelotaDX
    if pelotaY < 5 or pelotaY > 595:
        pelotaDY = -pelotaDY
    pelotaX += pelotaDX
    pelotaY += pelotaDY

    teclasPulsadas = pygame.key.get_pressed()
    if teclasPulsadas[K_a]:
        raquetaY += raquetaDY
    if teclasPulsadas[K_q]:
        raquetaY -= raquetaDY

    visor.fill((0,0,0))
    pygame.draw.circle(visor, BLANCO, (pelotaX,pelotaY),4,0)
    pygame.draw.rect(visor, BLANCO, (raquetaX,raquetaY,10,50))

    pygame.display.update()
```

Nuestra solución es mucho más simple y elegante. Pygame incluye una **función** que devuelve la **lista** de teclas que están pulsadas en un determinado momento. Su nombre es **pygame.key.get_pressed()**. Aprovechándonos de ello, almacenamos esa lista en una variable:

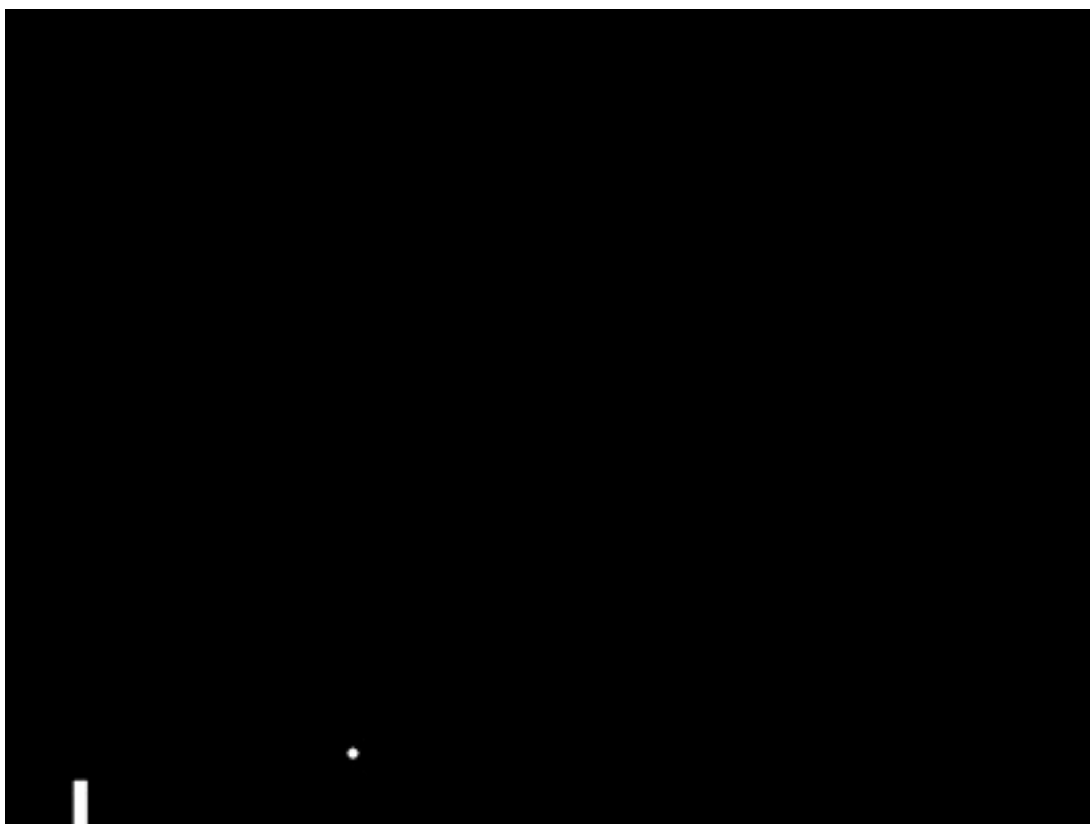
```
teclasPulsadas = pygame.key.get_pressed()
```

En realidad se trata de una **lista de booleanos**, con un valor **True** o **False** correspondiente para cada tecla del teclado, en función de que esté pulsada o no. Tal como está escrito Pygame, verificar si una tecla concreta está pulsada es muy fácil:

```
if teclasPulsadas[K_a]:  
    raquetaY += raquetaDY  
if teclasPulsadas[K_q]:  
    raquetaY -= raquetaDY
```

Y como estamos fuera del bucle de eventos, nada más fácil que modificar la posición de la raqueta según lo que esté pulsado en el momento de dibujarla; si está pulsada la tecla a, la raqueta se mueve hacia abajo. Y si está pulsada la tecla q, se mueve hacia arriba. Por supuesto, si no está pulsada ninguna de las dos, no se modifica la posición...

Ejecuta el programa y verifica el movimiento correcto con el teclado. Así da gusto. ¿Todo está en orden? Hmm... Aún falta un pequeño detalle...



pong08.py

En efecto; la raqueta se nos sale de la pantalla si pulsamos lo suficiente las teclas de movimiento. Y es el propio programa el que se debe de encargar de controlarlo y no mover la raqueta más allá de los límites establecidos para el juego.

Afortunadamente, la solución es fácil:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# pong08.py
# Movimiento definitivo de la raqueta

import pygame, sys
from pygame.locals import *

pygame.init()

fps = 60
tiempo = 0
BLANCO = (255,255,255)
pelotaX = 50
pelotaY = 50
pelotaDX = 5
pelotaDY = 5
raquetaX = 50
raquetaY = 250
raquetaDY = 5

visor = pygame.display.set_mode((800,600),FULLSCREEN)

while True:
    if pygame.time.get_ticks()-tiempo < 1000/fps:
        continue
    tiempo = pygame.time.get_ticks()
    for evento in pygame.event.get():
        if evento.type == KEYDOWN:
            if evento.key == K_ESCAPE:
                pygame.quit()
                sys.exit()
            if pelotaX < 5 or pelotaX > 795:
                pelotaDX = -pelotaDX
            if pelotaY < 5 or pelotaY > 595:
                pelotaDY = -pelotaDY
            pelotaX += pelotaDX
            pelotaY += pelotaDY
            teclasPulsadas = pygame.key.get_pressed()
            if teclasPulsadas[K_a]:
                raquetaY += raquetaDY
            if teclasPulsadas[K_q]:
```

```
raquetaY -= raquetaDY

if raquetaY < 0:
    raquetaY = 0
elif raquetaY > 550:
    raquetaY = 550

visor.fill((0,0,0))

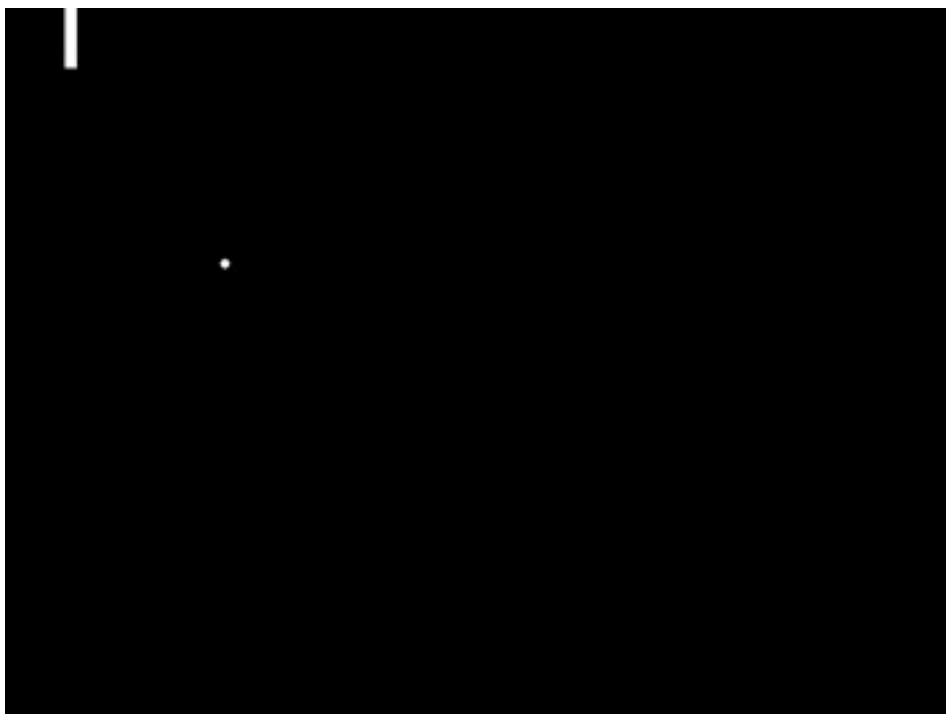
pygame.draw.circle(visor, BLANCO, (pelotaX,pelotaY),4,0)
pygame.draw.rect(visor, BLANCO, (raquetaX,raquetaY,10,50))

pygame.display.update()
```

Al mover la raqueta hay que verificar que no haya subido o bajado demasiado. Eso es lo que hacemos con

```
if raquetaY < 0:
    raquetaY = 0
elif raquetaY > 550:
    raquetaY = 550
```

(la diferencia está en que **raquetaY** marca la coordenada del extremo superior del rectángulo de la raqueta). En caso necesario, se corrige la posición para no salirse.



pong09.py

Es el turno de añadir al segundo jugador y, por lo tanto, la segunda raqueta. Observa:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# pong09.py
# Las dos raquetas con su movimiento

import pygame, sys
from pygame.locals import *

pygame.init()

fps = 60
tiempo = 0
BLANCO = (255,255,255)
pelotaX = 50
pelotaY = 50
pelotaDX = 5
pelotaDY = 5
raquetaX = 50
raquetaY = 250
raquetaDY = 5
raqueta2X = 740
raqueta2Y = 250
raqueta2DY = 5

visor = pygame.display.set_mode((800,600),FULLSCREEN)

while True:
    if pygame.time.get_ticks()-tiempo < 1000/fps:
        continue
    tiempo = pygame.time.get_ticks()
    for evento in pygame.event.get():
        if evento.type == KEYDOWN:
            if evento.key == K_ESCAPE:
                pygame.quit()
                sys.exit()
    if pelotaX < 5 or pelotaX > 795:
        pelotaDX = -pelotaDX
    if pelotaY < 5 or pelotaY > 595:
        pelotaDY = -pelotaDY
    pelotaX += pelotaDX
    pelotaY += pelotaDY
    teclasPulsadas = pygame.key.get_pressed()
    if teclasPulsadas[K_a]:
        raquetaY += raquetaDY
    if teclasPulsadas[K_q]:
        raquetaY -= raquetaDY
```

```

if raquetaY < 0:
    raquetaY = 0
elif raquetaY > 550:
    raquetaY = 550

if teclasPulsadas[K_l]:
    raqueta2Y += raqueta2DY
if teclasPulsadas[K_p]:
    raqueta2Y -= raqueta2DY
if raqueta2Y < 0:
    raqueta2Y = 0
elif raqueta2Y > 550:
    raqueta2Y = 550

visor.fill((0,0,0))

pygame.draw.circle(visor, BLANCO, (pelotaX,pelotaY),4,0)
pygame.draw.rect(visor, BLANCO, (raquetaX,raquetaY,10,50))
pygame.draw.rect(visor, BLANCO, (raqueta2X,raqueta2Y,10,50))

pygame.display.update()

```

Las modificaciones no son difíciles de entender, simplemente hay que añadir un código similar a la de la primera raqueta, teniendo en cuenta que las coordenadas y las teclas de movimiento son ahora distintas.

Para empezar, creamos las variables correspondientes a la posición y movimiento de esta segunda raqueta:

```

raqueta2X = 740
raqueta2Y = 250
raqueta2DY = 5

```

(fíjate que las variables tienen los mismos nombres que los de la otra raqueta, pero añadiendo un 2). Como la nueva raqueta está a la derecha de la pantalla, para dejar el mismo margen que la otra la coordenada **raqueta2X** ha de valer 740.

Ahora hay que mirar si están pulsadas la **tecla l** y la **tecla p**, ambas culpables del movimiento del segundo jugador:

```

if teclasPulsadas[K_l]:
    raqueta2Y += raqueta2DY
if teclasPulsadas[K_p]:
    raqueta2Y -= raqueta2DY

```

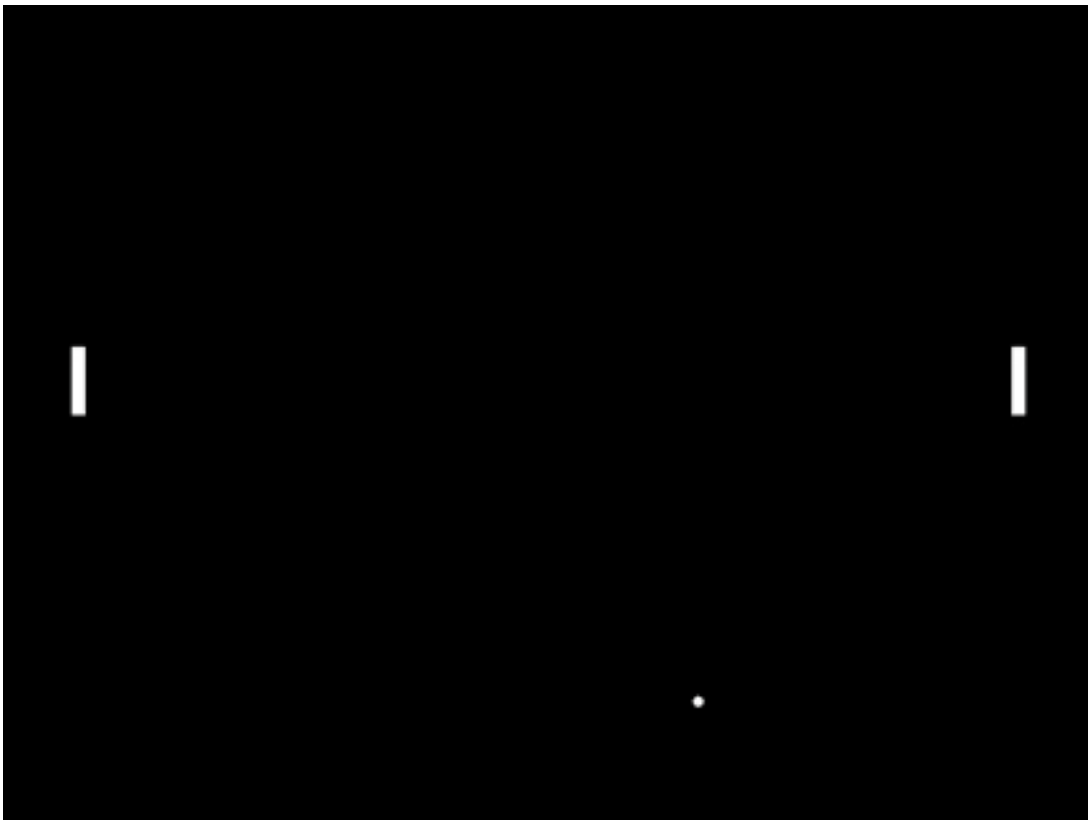
Y, de forma análoga a la primera raqueta, hay que vigilar que al moverla no se salga de los bordes de la pantalla:

```
if raqueta2Y < 0:  
    raqueta2Y = 0  
elif raqueta2Y > 550:  
    raqueta2Y = 550
```

Sólo queda dibujar de forma efectiva la raqueta del jugador 2 en su posición...

```
pygame.draw.rect(visor, BLANCO, (raqueta2X, raqueta2Y, 10, 50))
```

El resultado final, una vez ejecutado, ya va pareciéndose más a un juego de verdad:



¿Qué es lo próximo? ¡Claro! Falta que la pelota rebote en las raquetas, ya que ahora simplemente las atraviesa...

pong10.py

Para conseguir que la pelota rebote en las dos raquetas habrá que comparar su posición con la de ellas y, en caso necesario, invertir su movimiento como antes hacíamos con los bordes de la pantalla. Una observación; sólo hemos de cambiar el movimiento en el sentido horizontal.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# pong10.py
# La pelota rebota en las raquetas

import pygame, sys
from pygame.locals import *

pygame.init()

fps = 60
tiempo = 0
BLANCO = (255,255,255)
pelotaX = 50
pelotaY = 50
pelotaDX = 5
pelotaDY = 5
raquetaX = 50
raquetaY = 250
raquetaDY = 5
raqueta2X = 740
raqueta2Y = 250
raqueta2DY = 5

visor = pygame.display.set_mode((800,600),FULLSCREEN)

while True:
    if pygame.time.get_ticks()-tiempo < 1000/fps:
        continue
    tiempo = pygame.time.get_ticks()
    for evento in pygame.event.get():
        if evento.type == KEYDOWN:
            if evento.key == K_ESCAPE:
                pygame.quit()
                sys.exit()
    diff1 = pelotaY-raquetaY
    if pelotaX == raquetaX + 10 and diff1 >=0 and diff1 <= 50:
        pelotaDX = -pelotaDX
    diff2 = pelotaY-raqueta2Y
    if pelotaX == raqueta2X and diff2 >= 0 and diff2 <= 50:
        pelotaDX = -pelotaDX
    if pelotaY < 5 or pelotaY > 595:
        pelotaDY = -pelotaDY
```

```
pelotaX += pelotaDX
pelotaY += pelotaDY

teclasPulsadas = pygame.key.get_pressed()

if teclasPulsadas[K_a]:
    raquetaY += raquetaDY
if teclasPulsadas[K_q]:
    raquetaY -= raquetaDY

if raquetaY < 0:
    raquetaY = 0
elif raquetaY > 550:
    raquetaY = 550

if teclasPulsadas[K_l]:
    raqueta2Y += raqueta2DY
if teclasPulsadas[K_p]:
    raqueta2Y -= raqueta2DY

if raqueta2Y < 0:
    raqueta2Y = 0
elif raqueta2Y > 550:
    raqueta2Y = 550

visor.fill((0,0,0))

pygame.draw.circle(visor, BLANCO, (pelotaX,pelotaY),4,0)
pygame.draw.rect(visor, BLANCO, (raquetaX,raquetaY,10,50))
pygame.draw.rect(visor, BLANCO, (raqueta2X,raqueta2Y,10,50))

pygame.display.update()
```

Observa que hemos sustituido la comprobación de los programas anteriores sobre si la pelota alcanzaba los bordes izquierdo o derecho de la pantalla por un nuevo código.

Vamos por partes. Para saber si la pelota ha alcanzado una raqueta, necesitamos comparar sus **coordenadas X** (lo que es fácil) y sus **coordenadas Y** (lo que es algo más complicado, pues la pelota puede rebotar a lo largo de toda la raqueta). Para ello definamos la variable **diff1** (ponemos el 1 por que primero nos dedicamos a la primera raqueta) como la diferencia entre las **coordenadas Y** de la raqueta y la pelota.

```
diff1 = pelotaY-raquetaY
```

La raqueta mide 50 píxeles de alta, así que si esa diferencia es menor de 50, sabremos que la pelota habrá impactado correctamente y deberá rebotar. Eso sí, habrá que comprobar también que esa diferencia sea positiva, pues si no, significaría que la pelota va por arriba de la raqueta a menos de 50 píxeles, y entonces no hay impacto.

En cualquier caso, la comprobación (junto con la correspondiente a la **coordenada X**) quedará así:

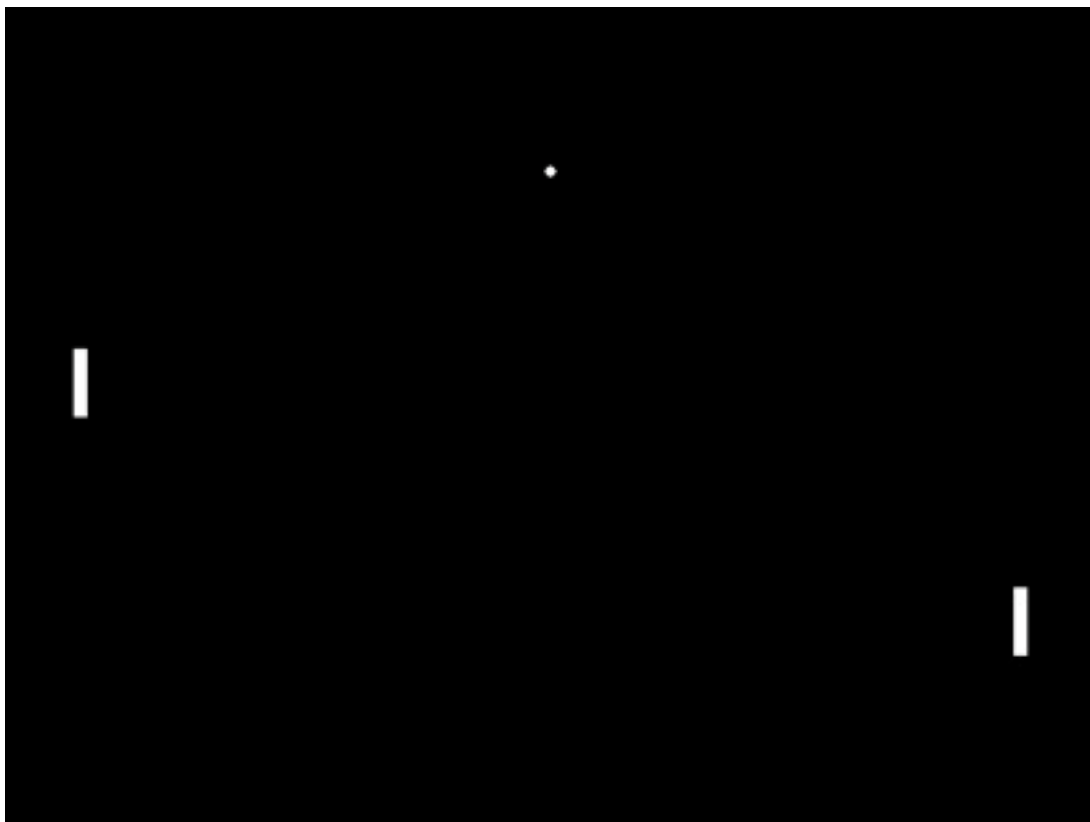
```
if pelotaX == raquetaX + 10 and diff1 >= 0 and diff1 <= 50:  
    pelotaDX = -pelotaDX
```

Obsérvala detenidamente. Como la raqueta izquierda tiene una anchura de 10 píxeles y la pelota viene de la derecha, el impacto se producirá cuando la **coordenada X** de la pelota sea 10 píxeles mayor que la de la raqueta, y además hemos de asegurarnos (como hemos comentado en el párrafo anterior) que se produce a la altura adecuada. En el caso de que todo eso ocurra, la pelota se hace rebotar cambiando el signo de su **pelotaDX**.

El código correspondiente al impacto con la raqueta de la derecha es muy similar y es prácticamente autoexplicativo.

```
diff2 = pelotaY-raqueta2Y  
if pelotaX == raqueta2X and diff2 >= 0 and diff2 <= 50:  
    pelotaDX = -pelotaDX
```

¡Nuestro juego ya es jugable! Pero si no lo haces bien y no le das a la pelota con la raqueta, ésta desaparece de la pantalla para siempre...



pong11.py

Cuando la pelota sale por la pantalla por que el jugador no le ha dado, el juego debe comenzar otra vez recolocando la pelota por el centro. Realmente esto no es complicado, simplemente debemos controlar cuando la coordenada X de la pelota es mayor que 800 o menor que 0 y cambiarla en ese momento.

Por otra parte, si hacemos simplemente eso, la pelota seguirá moviéndose en la misma dirección, acosando al jugador que acaba de fallar. Es, pues, conveniente invertir la dirección de la pelota y hacerlo pasado un corto lapso de tiempo para que los jugadores estén dispuestos a volver a jugar:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# pong11.py
# El juego continúa al fallar

import pygame, sys
import time
from pygame.locals import *
pygame.init()

fps = 60
tiempo = 0
BLANCO = (255,255,255)
pelotaX = 50
pelotaY = 50
pelotaDX = 5
pelotaDY = 5
raquetaX = 50
raquetaY = 250
raquetaDY = 5
raqueta2X = 740
raqueta2Y = 250
raqueta2DY = 5

visor = pygame.display.set_mode((800,600),FULLSCREEN)

while True:
    if pygame.time.get_ticks()-tiempo < 1000/fps:
        continue
    tiempo = pygame.time.get_ticks()
    for evento in pygame.event.get():
        if evento.type == KEYDOWN:
            if evento.key == K_ESCAPE:
                pygame.quit()
                sys.exit()
    diff1 = pelotaY-raquetaY
    if pelotaX == raquetaX + 10 and diff1 >= 0 and diff1 <= 50:
        pelotaDX = -pelotaDX
    diff2 = pelotaY-raqueta2Y
```

```
if pelotaX == raqueta2X and diff2 >= 0 and diff2 <= 50:
    pelotaDX = -pelotaDX

if pelotaY < 5 or pelotaY > 595:
    pelotaDY = -pelotaDY

pelotaX += pelotaDX
pelotaY += pelotaDY

teclasPulsadas = pygame.key.get_pressed()

if teclasPulsadas[K_a]:
    raquetaY += raquetaDY
if teclasPulsadas[K_q]:
    raquetaY -= raquetaDY
if raquetaY < 0:
    raquetaY = 0
elif raquetaY > 550:
    raquetaY = 550

if teclasPulsadas[K_l]:
    raqueta2Y += raqueta2DY
if teclasPulsadas[K_p]:
    raqueta2Y -= raqueta2DY
if raqueta2Y < 0:
    raqueta2Y = 0
elif raqueta2Y > 550:
    raqueta2Y = 550

if pelotaX > 800 or pelotaX < 0:
    time.sleep(1)
    pelotaX = 400
    pelotaDX = -pelotaDX
    raquetaY = 250
    raqueta2Y = 250

visor.fill((0,0,0))

pygame.draw.circle(visor, BLANCO, (pelotaX,pelotaY),4,0)
pygame.draw.rect(visor, BLANCO, (raquetaX,raquetaY,10,50))
pygame.draw.rect(visor, BLANCO, (raqueta2X,raqueta2Y,10,50))

pygame.display.update()
```

Para empezar, para poder implementar la pausa al fallar el jugador, necesitamos importar el **módulo time** que contiene a la función **sleep** que hace precisamente eso:

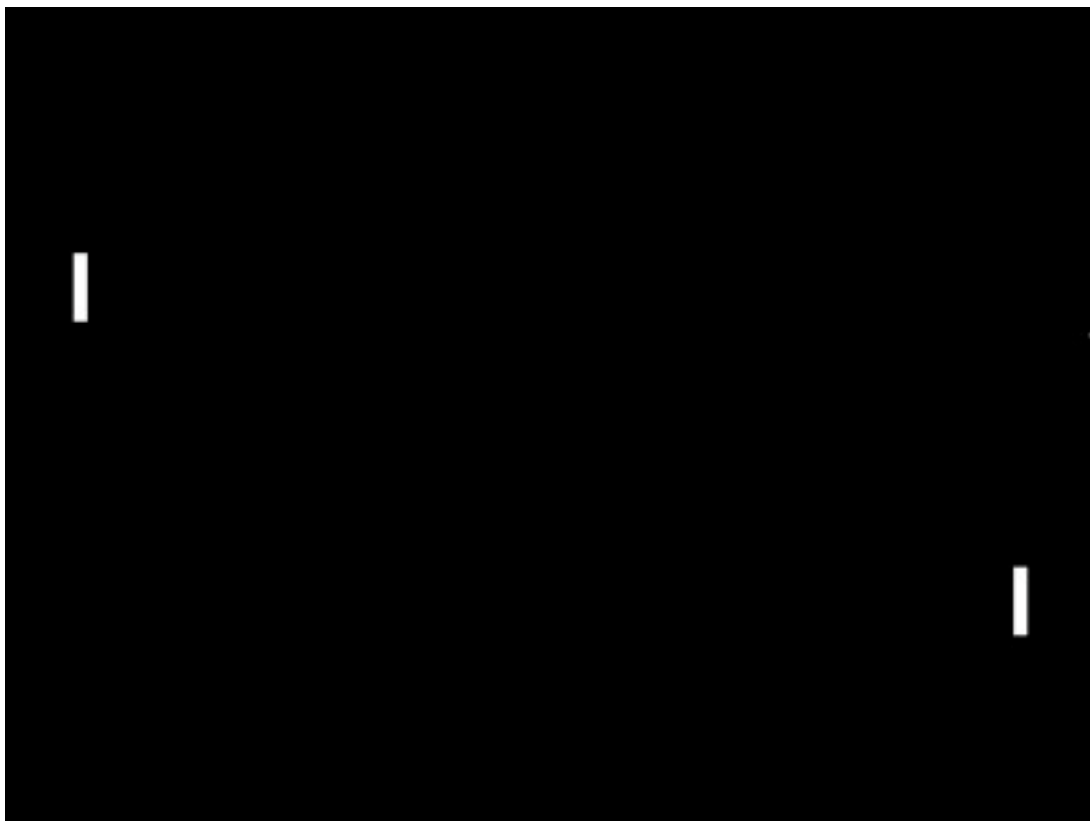
```
import time
```

Y ahora le toca al código que se encarga de volver a poner la pelota en juego:

```
if pelotaX > 800 or pelotaX < 0:  
    time.sleep(1)  
    pelotaX = 400  
    pelotaDX = -pelotaDX  
    raquetaY = 250  
    raqueta2Y = 250
```

Primero miramos con el if si la pelota abandona la pantalla (es decir, si se ha fallado). En caso afirmativo, y por orden, se hace una pausa de 1 segundo, se recoloca la pelota en el centro, se invierte su movimiento y sitúan las raquetas de los jugadores en sus posiciones iniciales.

Pruébalo y verás que funciona. Ni que decir tiene que se puede hacer de otras formas, algunas mejores incluso, pero ésta nos vale para nuestros propósitos.



pong12.py

¡Nuestro juego es un juego de competición!. Debemos, por lo tanto, llevar la cuenta de los errores de cada jugador. Cuando uno de los dos falla, el contrario gana un punto. Veamos como llevar esa cuenta:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

# pong12.py
# Puntuando a los jugadores

import pygame, sys
import time
from pygame.locals import *

pygame.init()

fps = 60
tiempo = 0
BLANCO = (255,255,255)
pelotaX = 50
pelotaY = 50
pelotaDX = 5
pelotaDY = 5
raquetaX = 50
raquetaY = 250
raquetaDY = 5
raqueta2X = 740
raqueta2Y = 250
raqueta2DY = 5
puntos1 = 0
puntos2 = 0

visor = pygame.display.set_mode((800,600),FULLSCREEN)

while True:
    if pygame.time.get_ticks()-tiempo < 1000/fps:
        continue
    tiempo = pygame.time.get_ticks()
    for evento in pygame.event.get():
        if evento.type == KEYDOWN:
            if evento.key == K_ESCAPE:
                pygame.quit()
                sys.exit()
    diff1 = pelotaY-raquetaY
    if pelotaX == raquetaX + 10 and diff1 >= 0 and diff1 <= 50:
        pelotaDX = -pelotaDX
    diff2 = pelotaY-raqueta2Y
    if pelotaX == raqueta2X and diff2 >= 0 and diff2 <= 50:
```

```

    pelotaDX = -pelotaDX

    if pelotaY < 5 or pelotaY > 595:
        pelotaDY = -pelotaDY

    pelotaX += pelotaDX
    pelotaY += pelotaDY
    teclasPulsadas = pygame.key.get_pressed()
    if teclasPulsadas[K_a]:
        raquetaY += raquetaDY
    if teclasPulsadas[K_q]:
        raquetaY -= raquetaDY
    if raquetaY < 0:
        raquetaY = 0
    elif raquetaY > 550:
        raquetaY = 550
    if teclasPulsadas[K_l]:
        raqueta2Y += raqueta2DY
    if teclasPulsadas[K_p]:
        raqueta2Y -= raqueta2DY
    if raqueta2Y < 0:
        raqueta2Y = 0
    elif raqueta2Y > 550:
        raqueta2Y = 550

    if pelotaX > 800 or pelotaX < 0:
        time.sleep(1)
        raquetaY = 250
        raqueta2Y = 250
        if pelotaX > 800:
            puntos1 = puntos1 + 1
        else:
            puntos2 = puntos2 + 1
        print 'Jugador1:',puntos1,'Jugador2:',puntos2
        pelotaX = 400
        pelotaDX = -pelotaDX

visor.fill((0,0,0))

pygame.draw.circle(visor, BLANCO, (pelotaX,pelotaY),4,0)
pygame.draw.rect(visor, BLANCO, (raquetaX,raquetaY,10,50))
pygame.draw.rect(visor, BLANCO, (raqueta2X,raqueta2Y,10,50))

pygame.display.update()

```

Por descontado, tenemos que definir dos variables que almacenen los puntos que llevan ganados cada jugador:

```

puntos1 = 0
puntos2 = 0

```

Cuando la pelota alcanza el lado derecho de la pantalla (es decir, su **coordenada X** llega a los 800 píxeles) es por que ha fallado el segundo jugador y, por lo tanto, hay que sumar un punto al marcador del primer jugador. Y lo contrario ocurre en el otro extremo de la pantalla:

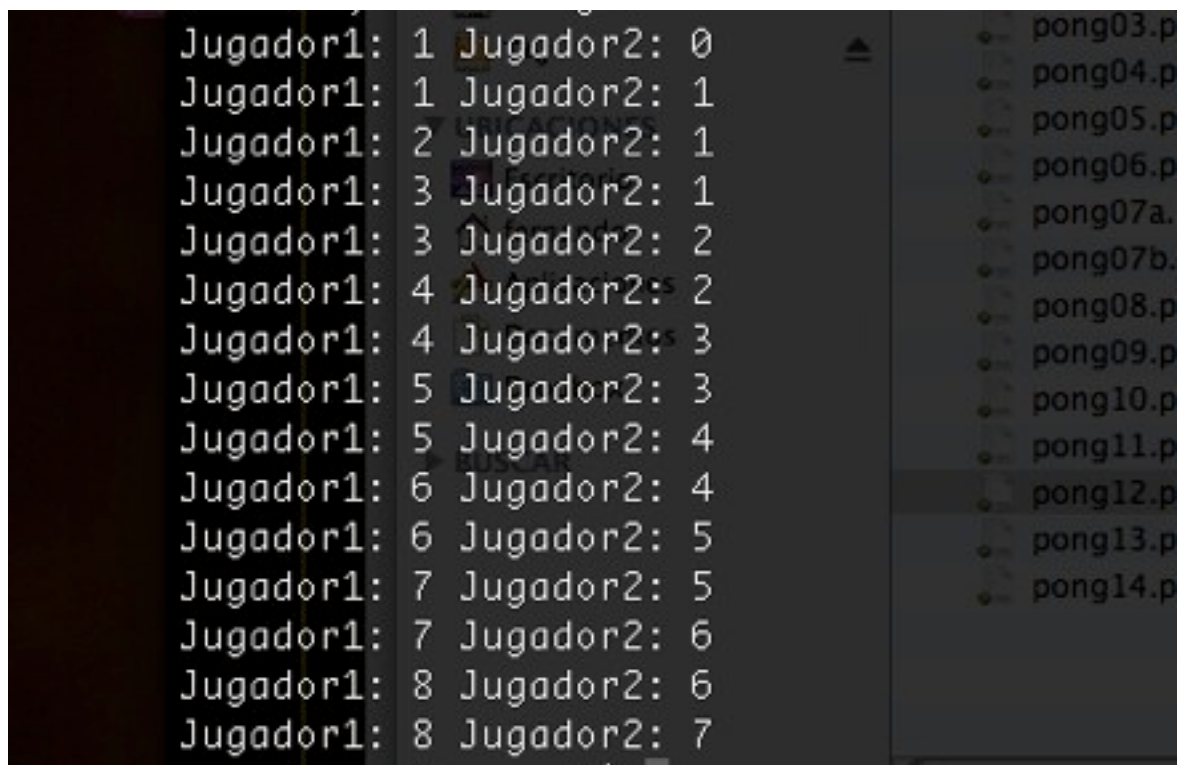
```
if pelotaX > 800:  
    puntos1 = puntos1 + 1  
else:  
    puntos2 = puntos2 + 1
```

Fíjate que este código lo hemos puesto dentro del **if** que se encarga de detectar si la pelota ha alcanzado uno de los bordes de la pantalla. Por eso, ya sólo tenemos que distinguir si ha ocurrido con por el lado derecho o por el izquierdo, y de ahí la comparación.

Finalmente, para verificar que todo está en orden, le decimos a Python que escriba los valores de las puntuaciones (de momento, lo mostrará simplemente en la línea de comandos, no en la pantalla de juego):

```
print 'Jugador1:',puntos1,'Jugador2:',puntos2
```

Si ejecutamos ahora el programa, aparentemente no hay ningún cambio; sin embargo, si miramos la terminal (o la ventana que nos muestra la información de la ejecución) veremos los tanteos escritos progresivamente:



```
Jugador1: 1 Jugador2: 0  
Jugador1: 1 Jugador2: 1  
Jugador1: 2 Jugador2: 1  
Jugador1: 3 Jugador2: 1  
Jugador1: 3 Jugador2: 2  
Jugador1: 4 Jugador2: 2  
Jugador1: 4 Jugador2: 3  
Jugador1: 5 Jugador2: 3  
Jugador1: 5 Jugador2: 4  
Jugador1: 6 Jugador2: 4  
Jugador1: 6 Jugador2: 5  
Jugador1: 7 Jugador2: 5  
Jugador1: 7 Jugador2: 6  
Jugador1: 8 Jugador2: 6  
Jugador1: 8 Jugador2: 7
```

pong13.py

Es el turno de poner un marcador de juego en condiciones. Si te fijas en la imagen original del juego (al principio de este documento), verás que también debemos poner unas líneas que representen la red de medio campo. Lo podemos hacer de la siguiente manera:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# pong013.py
# El Marcador

import pygame, sys
import time
from pygame.locals import *

pygame.init()

fps = 60
tiempo = 0
BLANCO = (255,255,255)
pelotaX = 50
pelotaY = 50
pelotaDX = 5
pelotaDY = 5
raquetaX = 50
raquetaY = 250
raquetaDY = 5
raqueta2X = 740
raqueta2Y = 250
raqueta2DY = 5
puntos1 = 0
puntos2 = 0

pygame.init()
tipoLetra = pygame.font.SysFont('arial', 96)
visor = pygame.display.set_mode((800,600),FULLSCREEN)

while True:
    if pygame.time.get_ticks()-tiempo < 1000/fps:
        continue
    tiempo = pygame.time.get_ticks()
    for evento in pygame.event.get():
        if evento.type == KEYDOWN:
            if evento.key == K_ESCAPE:
                pygame.quit()
                sys.exit()

    diff1 = pelotaY-raquetaY
    if pelotaX == raquetaX + 10 and diff1 >= 0 and diff1 <= 50:
        pelotaDX = -pelotaDX
```

```
diff2 = pelotaY-raqueta2Y
if pelotaX == raqueta2X and diff2 >= 0 and diff2 <= 50:
    pelotaDX = -pelotaDX
if pelotaY < 5 or pelotaY > 595:
    pelotaDY = -pelotaDY
pelotaX += pelotaDX
pelotaY += pelotaDY
teclasPulsadas = pygame.key.get_pressed()
if teclasPulsadas[K_a]:
    raquetaY += raquetaDY
if teclasPulsadas[K_q]:
    raquetaY -= raquetaDY
if raquetaY < 0:
    raquetaY = 0
elif raquetaY > 550:
    raquetaY = 550
if teclasPulsadas[K_l]:
    raqueta2Y += raqueta2DY
if teclasPulsadas[K_p]:
    raqueta2Y -= raqueta2DY
if raqueta2Y < 0:
    raqueta2Y = 0
elif raqueta2Y > 550:
    raqueta2Y = 550
if pelotaX > 800 or pelotaX < 0:
    time.sleep(1)
    raquetaY = 250
    raqueta2Y = 250
    if pelotaX > 800:
        puntos1 = puntos1 + 1
    else:
        puntos2 = puntos2 + 1
    pelotaX = 400
    pelotaDX = -pelotaDX

visor.fill((0,0,0))

pygame.draw.circle(visor, BLANCO, (pelotaX,pelotaY),4,0)
pygame.draw.rect(visor, BLANCO, (raquetaX,raquetaY,10,50))
pygame.draw.rect(visor, BLANCO, (raqueta2X,raqueta2Y,10,50))

for i in range(10):
    pygame.draw.rect(visor, BLANCO, (398,10+60*i,4,30))

marcador1 = tipoLetra.render(str(puntos1), True, BLANCO)
marcador2 = tipoLetra.render(str(puntos2), True, BLANCO)
visor.blit(marcador1, (300,20,50,50))
visor.blit(marcador2, (450,20,50,50))

pygame.display.update()
```


Observa que, de paso, hemos eliminado las líneas de la versión del programa anterior que escribían la puntuación de los jugadores en la terminal.

Vamos por partes; empecemos por el dibujo de la red. Se trata de una línea discontinua vertical a mitad de pantalla. Podemos, desde luego, dibujar cada uno de los segmentos a mano. Pero si usamos algo de matemáticas, podemos hacerlo de un tirón:

```
for i in range(10):  
    pygame.draw.rect(visor, BLANCO, (398,10+60*i,4,30))
```

range(10) nos devuelve, como ya sabemos, la **lista** [0,1,2,3,4,5,6,7,8,9]. Con el bucle **for**, lo que conseguimos es que para cada uno de los valores de esa lista (los valores de **i**) se dibuje un pequeño rectángulo de 4 píxeles de anchura y 30 de altura. ¿Dónde se dibujan? La **coordenada X** siempre es 398 (para que quede centrado en la mitad, 400) y la **coordenada Y** va aumentando con los valores de **i**. Muchas veces, cuando programas algo así, haces las cuentas a ojo y luego ejecutas el programa y miras el resultado. Si no queda como quieres, modificas los valores convenientemente y así sucesivamente.

Trabajemos ahora con el marcador de puntos. Como es un texto (de hecho, un texto que contiene números) lo que vamos a escribir, primero vamos a definir de qué tipo es éste:

```
tipoLetra = pygame.font.SysFont('arial', 96)
```

Hemos elegido el tipo de letra 'arial' y un tamaño grande, 96 puntos. Recuerda que para ver qué tipos de letra puedes usar dispones de la función **pygame.font.get_fonts()**, pero ten en cuenta que el usuario final no tiene por qué tener instaladas en su sistema los mismos tipos de letra que tú.

Lo siguiente es generar el texto correspondiente a los puntos de cada jugador:

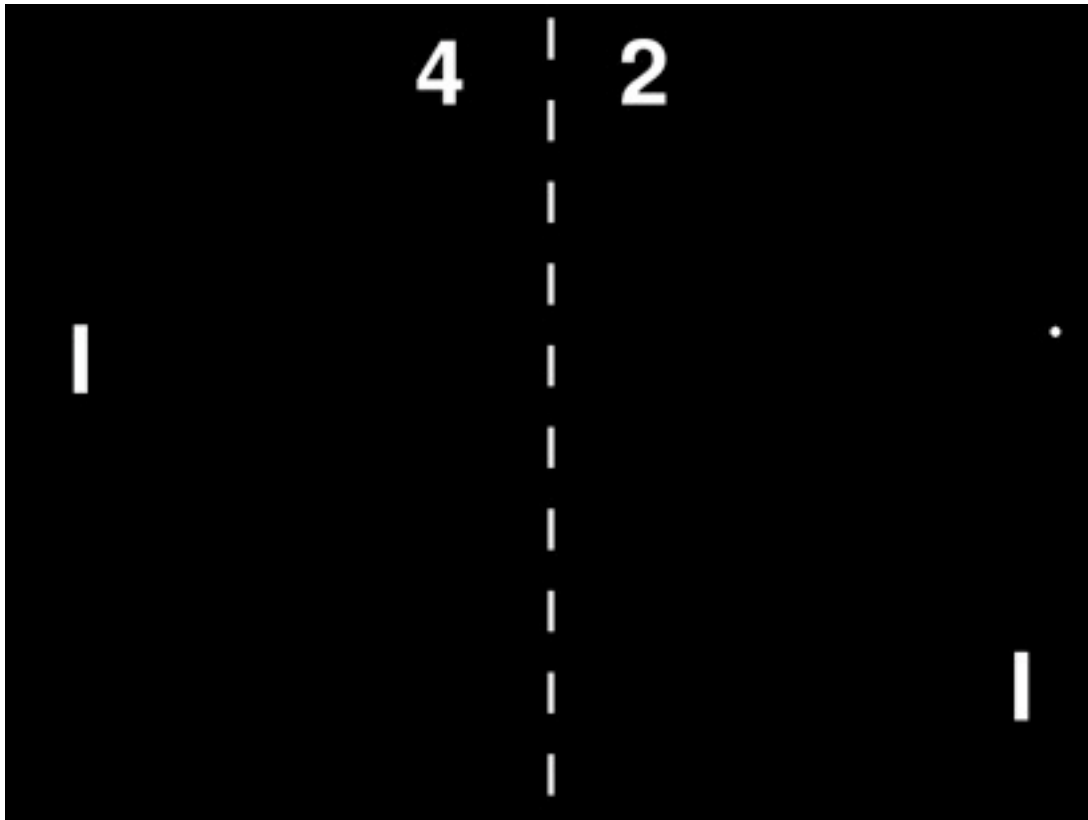
```
marcador1 = tipoLetra.render(str(puntos1), True, BLANCO)  
marcador2 = tipoLetra.render(str(puntos2), True, BLANCO)
```

Sólo queda situarlos en el lugar adecuado...

```
visor.blit(marcador1, (300,20,50,50))  
visor.blit(marcador2, (450,20,50,50))
```

Situar un texto, para que quede bien centrado y perfecto, lleva algo más de trabajo. Pero a nuestros efectos, esto nos bastará. A la función **blit** debemos pasarle un **rect** para decirle dónde debe dibujar; recuerda que es una **tupla** de 4 números, los dos primeros son las **coordenadas X** e **Y** de la esquina superior de la zona rectangular que queremos definir, y los dos siguientes son la **anchura** y la **altura**. Nuestro texto (los puntos de cada jugador) quedarán encajados en dicha zona.

Ejecuta el programa. ¡Queda bonito! ¿Entiendes de paso por qué situar el texto es algo más laborioso? Dependiendo de lo que ocupa, los números quedan equidistantes a la red o no. Un buen programador, debe tener en cuenta esos detalles. También es cierto que el trabajo usual suele ser publicar actualizaciones en las que se van mejorando las características o se corrigen errores. Así es el mercado libre...



Nos queda pulir algunos detalles. Por ejemplo, evitar que el juego empiece automáticamente al arrancarlo. De hecho, según cómo sea tu ordenador notarás que el cambio de resolución de pantalla es lento y, cuando se ha hecho, el juego ya está en marcha (esto pasa con algunos tipos de monitores planos; puede incluso que la pantalla de juego no la ajusten bien del todo). También hay que terminar el juego a una cierta puntuación (21 suele ser el tanteo habitual). De eso nos vamos a encargar en la siguiente versión.

pong14.py

Vamos a implementar el arranque del juego:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# pong14.py
# Implementando el arranque del programa

import pygame, sys
import time
from pygame.locals import *

pygame.init()

fps = 60
tiempo = 0
BLANCO = (255,255,255)
pelotaX = 50
pelotaY = 50
pelotaDX = 5
pelotaDY = 5
raquetaX = 50
raquetaY = 250
raquetaDY = 5
raqueta2X = 740
raqueta2Y = 250
raqueta2DY = 5
puntos1 = 0
puntos2 = 0
esperar = True

tipoLetra = pygame.font.SysFont('arial', 96)
tipoLetra2 = pygame.font.SysFont('arial', 24)

visor = pygame.display.set_mode((800,600),FULLSCREEN)

mensaje1 = 'P O N G'
texto1 = tipoLetra.render(mensaje1, True, BLANCO)
mensaje2 = 'Pulsa una tecla para comenzar'
texto2 = tipoLetra2.render(mensaje2, True, BLANCO)
visor.blit(texto1, (50,250,200,100))
visor.blit(texto2, (400,292,350,30))
pygame.display.update()

while esperar:
    for evento in pygame.event.get():
        if evento.type == KEYDOWN:
            esperar = False
```

```
while True:
    if pygame.time.get_ticks()-tiempo < 1000/fps:
        continue
    tiempo = pygame.time.get_ticks()

    for evento in pygame.event.get():

        if evento.type == KEYDOWN:
            if evento.key == K_ESCAPE:
                pygame.quit()
                sys.exit()

        diff1 = pelotaY-raquetaY
        if pelotaX == raquetaX + 10 and diff1 >= 0 and diff1 <= 50:
            pelotaDX = -pelotaDX
        diff2 = pelotaY-raqueta2Y
        if pelotaX == raqueta2X and diff2 >= 0 and diff2 <= 50:
            pelotaDX = -pelotaDX

        if pelotaY < 5 or pelotaY > 595:
            pelotaDY = -pelotaDY

        pelotaX += pelotaDX
        pelotaY += pelotaDY
        teclasPulsadas = pygame.key.get_pressed()
        if teclasPulsadas[K_a]:
            raquetaY += raquetaDY
        if teclasPulsadas[K_q]:
            raquetaY -= raquetaDY
        if raquetaY < 0:
            raquetaY = 0
        elif raquetaY > 550:
            raquetaY = 550
        if teclasPulsadas[K_l]:
            raqueta2Y += raqueta2DY
        if teclasPulsadas[K_p]:
            raqueta2Y -= raqueta2DY
        if raqueta2Y < 0:
            raqueta2Y = 0
        elif raqueta2Y > 550:
            raqueta2Y = 550

        if pelotaX > 800 or pelotaX < 0:
            time.sleep(1)
            raquetaY = 250
            raqueta2Y = 250
            if pelotaX > 800:
                puntos1 = puntos1 + 1
            else:
                puntos2 = puntos2 + 1
```

```
pelotaX = 400
pelotaDX = -pelotaDX

visor.fill((0,0,0))

pygame.draw.circle(visor, BLANCO, (pelotaX,pelotaY),4,0)
pygame.draw.rect(visor, BLANCO, (raquetaX,raquetaY,10,50))
pygame.draw.rect(visor, BLANCO, (raqueta2X,raqueta2Y,10,50))
for i in range(10):
    pygame.draw.rect(visor, BLANCO, (398,10+60*i,4,30))
marcador1 = tipoLetra.render(str(puntos1), True, BLANCO)
marcador2 = tipoLetra.render(str(puntos2), True, BLANCO)
visor.blit(marcador1, (300,20,50,50))
visor.blit(marcador2, (450,20,50,50))

pygame.display.update()
```

La idea es mostrar en pantalla el siguiente mensaje



y esperar a que el usuario pulse una tecla para empezar a jugar.

Un detalle importante es que cuando no tienes que mostrar una animación si no simplemente un texto estático, no hace falta que pongas el dibujado dentro de un bucle para que haga la tarea repetidamente; basta hacerlo una vez y esperar el tiempo que se necesite. Por ello, el código necesario lo hemos puesto sin más antes del bucle de juego.

Fíjate que hay dos tipos de texto. Las letras de PONG tienen un tamaño grande y usamos el mismo tipo que el de los marcadores de tanteo. Las letras del mensaje para que el jugador pulse una tecla son, por otra parte, más pequeñas:

```
tipoLetra2 = pygame.font.SysFont('arial', 24)
```

Ahora simplemente hay que seguir los mismos pasos que con los marcadores; generar el texto, situarlo y, esta vez sí, volcarlo en pantalla para que se muestre antes del juego:

```
mensaje1 = 'P O N G'
texto1 = tipoLetra.render(mensaje1, True, BLANCO)
mensaje2 = 'Pulsa una tecla para comenzar'
texto2 = tipoLetra2.render(mensaje2, True, BLANCO)
visor.blit(texto1, (50,250,200,100))
visor.blit(texto2, (400,292,350,30))
pygame.display.update()
```

Las coordenadas y tamaños de las zonas donde situarlos ya es una cuestión de elección; puedes elegir las que más te gusten.

Bien. Ahora, y antes de que empiece el juego, debemos esperar a que se pulse una tecla. ¿Cómo hacerlo? En cierto modo, ya podemos imaginar la solución; usemos un bucle de eventos y veamos cuando se produce un evento del tipo **KEYDOWN**, en cuyo caso se saldrá del bucle y se continuará con el juego. Con ese fin, vamos a usar una variable **booleana, esperar**, que inicialmente será **True**. Mientras mantenga ese valor, seguiremos en el bucle...

```
esperar = True
```

Pasemos ahora al susodicho bucle; quedará claro el uso de la variable **esperar**:

```
while esperar:
    for evento in pygame.event.get():
        if evento.type == KEYDOWN:
            esperar = False
```

En efecto, como **esperar** es **True**, el bucle **while** se ejecutará una y otra vez. Pero cuando se produzca un evento de tipo **KEYDOWN** (es decir, se pulse una tecla), la variable **esperar** se cambia a **False** y el bucle ya no se efectuará más.

Pruébalo. Funciona estupendamente...

pong15.py

A medida que crece el código de un programa, crecen los problemas: Nuestra criatura empieza a ser muy grande, el código puede ser repetitivo (por ejemplo, es posible que, en algún que otro momento del programa, el jugador tenga que volver a esperar; ¿escribiremos de nuevo el código de antes? ¿Y si hay que hacerlo más veces?), podemos empezar a no recordar para qué hemos definido esto o por qué hemos puesto aquello...

Es tiempo de recapitulación. En esta nueva versión del programa no vamos a añadir nada nuevo, simplemente vamos a reorganizarlo todo de manera que quede más claro y reusable. De esta forma, será más fácil abordar las modificaciones finales que nos aguardan.

Veamos cómo nos queda:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

#-----#
# pong15.py
# Organizando el código
#-----#

#-----#
# Importación de los módulos necesarios
#-----#

import pygame, sys
import time
from pygame.locals import *

#-----#
# Inicialización de Pygame
#-----#

pygame.init()

#-----#
# Definición de variables
#-----#

fps = 60
tiempo = 0
BLANCO = (255,255,255)
pelotaX = 50
pelotaY = 50
pelotaDX = 5
pelotaDY = 5
raquetaX = 50
raquetaY = 250
raquetaDY = 5
```

```
raqueta2X = 740
raqueta2Y = 250
raqueta2DY = 5
puntos1 = 0
puntos2 = 0
tipoLetra = pygame.font.SysFont('arial', 96)
tipoLetra2 = pygame.font.SysFont('arial', 24)

#-----#
# Creación de la pantalla de juego (SURFACE)
#-----#

visor = pygame.display.set_mode((800,600),FULLSCREEN)

#-----#
# Funciones del programa
#-----#

def pausa():
    # Esta función hace que se espera hasta que se pulse una tecla
    esperar = True
    while esperar:
        for evento in pygame.event.get():
            if evento.type == KEYDOWN:
                esperar = False

def mostrarIntro():
    # Muestra la pantalla de inicio y espera
    mensaje1 = 'P O N G'
    texto1 = tipoLetra.render(mensaje1, True, BLANCO)
    mensaje2 = 'Pulsa una tecla para comenzar'
    texto2 = tipoLetra2.render(mensaje2, True, BLANCO)
    visor.blit(texto1, (50,250,200,100))
    visor.blit(texto2, (400,292,350,30))
    pygame.display.update()
    pausa()

def dibujarJuego():
    # Dibuja la mesa, la pelota, las raquetas y los marcadores
    # Primero borra la pantalla en negro
    visor.fill((0,0,0))
    # Dibuja la pelota y las raquetas
    pygame.draw.circle(visor, BLANCO, (pelotaX,pelotaY),4,0)
    pygame.draw.rect(visor, BLANCO, (raquetaX,raquetaY,10,50))
    pygame.draw.rect(visor, BLANCO, (raqueta2X,raqueta2Y,10,50))
    # Dibuja la red
    for i in range(10):
        pygame.draw.rect(visor, BLANCO, (398,10+60*i,4,30))
    # Dibuja los marcadores
    marcador1 = tipoLetra.render(str(puntos1), True, BLANCO)
```



```
marcador2 = tipoLetra.render(str(puntos2), True, BLANCO)
visor.blit(marcador1, (300,20,50,50))
visor.blit(marcador2, (450,20,50,50))
# Y, finalmente, lo vuelca todo en pantalla
pygame.display.update()

#-----#
# Cuerpo principal del juego
#-----#

mostrarIntro()

while True:

    #-----#
    # Gestionar la velocidad del juego
    #-----#

    if pygame.time.get_ticks()-tiempo < 1000/fps:
        continue
    tiempo = pygame.time.get_ticks()

    #-----#
    # Bucle de eventos: Mirar si se quiere terminar el juego
    #-----#

    for evento in pygame.event.get():
        if evento.type == KEYDOWN:
            if evento.key == K_ESCAPE:
                pygame.quit()
                sys.exit()

    #-----#
    # Mover la pelota
    #-----#

    # Primero hay que vigilar por si hay que cambiar de dirección
    # Mira si se impacta con el jugador 1
    diff1 = pelotaY-raquetaY
    if pelotaX == raquetaX + 10 and diff1 >= 0 and diff1 <= 50:
        pelotaDX = -pelotaDX
    # Mira si se impacta con el jugador 2
    diff2 = pelotaY-raqueta2Y
    if pelotaX == raqueta2X and diff2 >= 0 and diff2 <= 50:
        pelotaDX = -pelotaDX
    # Mira si se ha llegado al borde de la pantalla
    if pelotaY < 5 or pelotaY > 595:
        pelotaDY = -pelotaDY
    # Mueve la pelota
    pelotaX += pelotaDX
```

```
pelotaY += pelotaDY

#-----#
# Mover las raquetas
#-----#

# Mira si el jugador 1 mueve la raqueta
teclasPulsadas = pygame.key.get_pressed()
if teclasPulsadas[K_a]:
    raquetaY += raquetaDY
if teclasPulsadas[K_q]:
    raquetaY -= raquetaDY
# Vigilar que la raqueta no se salga de la pantalla
if raquetaY < 0:
    raquetaY = 0
elif raquetaY > 550:
    raquetaY = 550
# Ahora hacemos lo mismo con el jugador 2
if teclasPulsadas[K_l]:
    raqueta2Y += raqueta2DY
if teclasPulsadas[K_p]:
    raqueta2Y -= raqueta2DY
if raqueta2Y < 0:
    raqueta2Y = 0
elif raqueta2Y > 550:
    raqueta2Y = 550

#-----#
# Mirar si se ha ganado un punto
#-----#

# Primero, mira si la pelota ha llegado al borde
if pelotaX > 800 or pelotaX < 0:
    # En tal caso, recolocar jugadores y pelota y cambiar la puntuación
    time.sleep(1)
    raquetaY = 250
    raqueta2Y = 250
    if pelotaX > 800:
        puntos1 = puntos1 + 1
    else:
        puntos2 = puntos2 + 1
    pelotaX = 400
    pelotaDX = -pelotaDX

#-----#
# Dibujar el juego en pantalla
#-----#

dibujarJuego()
```

Unas pocas matizaciones:

Como puede verse, cuando el código es largo y para ayudar al programador a leerlo, las líneas de comentario se suelen aumentar y adornar para que sean fáciles de localizar. Fíjate cómo hemos separado las secciones con líneas horizontales y, los pequeños comentarios, simplemente los hemos colocado junto a su código.

Otra cosa que ayuda bastante, es agrupar las porciones de código que sirvan para una tarea en concreto o que se hagan (o se puedan hacer posteriormente) repetitivamente en funciones. Una vez que se colocan en el cuerpo de una función, realizar esa tarea sólo es invocar a la función. Así hemos definido **pausa()**, **mostrarIntro()** y **dibujarJuego()**. De hecho, **mostrarIntro()** llama a su vez a **pausa()**.

También es típico (y ya lo hemos visto), tener organizadas las diferentes partes del programa. Al principio del código (tras la línea que permite ejecutarlo desde una terminal y la que marca la codificación) se suele poner el **nombre del programa y un breve comentario**. A continuación se sitúa la sección de la **importación de módulos**. Detrás, la sección de **declaración de variables y constantes**. Y tras ella, la de las **definiciones de funciones**. Una vez terminado todo lo anterior, se puede proceder con el **flujo principal del programa**.

Hay que hacer varias advertencias con las funciones. La primera de ellas es que tengas cuidado de **no definir una función que usa una variable o una constante antes de que ésta se haya definido a su vez**. Éste es un error típico. Pero si eres ordenado y sigues la pauta indicada en el párrafo anterior, no debería ocurrirte.

La segunda advertencia, y más delicada, quizá tenga que ver con algo que hayas podido pensar. ¿Por qué no definir más funciones con mas secciones de código? Por ejemplo, ¿por qué no hemos usado una función para mover las raquetas? La razón está en las variables. Dentro de una función puede usarse una **variable numérica** que se haya definido fuera, pero no puede modificarse (esta restricción no ocurre con otros tipos de datos más complejos). Así que, como para mover las raquetas debemos modificar sus coordenadas, mejor no usar una función. Bien, en realidad esto no es cierto al 100% como sabes, pues existe la instrucción **global** que permite, al principio de una función, declarar una variable externa para que pueda modificarse. Pero no es una solución que se considere elegante...

pong16.py

Debemos terminar el juego cuando uno de los jugadores alcance los 11 puntos y, a continuación, dar la opción de volver a jugar. Con ello, el juego estará prácticamente operativo. Éste será el código:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

#-----#
# pong16.py
# Implementando el final de juego
#-----#

#-----#
# Importación de los módulos necesarios
#-----#

import pygame, sys
import time
from pygame.locals import *

#-----#
# Inicialización de Pygame
#-----#

pygame.init()

#-----#
# Definición de variables
#-----#

fps = 60
tiempo = 0
BLANCO = (255,255,255)
pelotaX = 50
pelotaY = 50
pelotaDX = 5
pelotaDY = 5
raquetaX = 50
raquetaY = 250
raquetaDY = 5
raqueta2X = 740
raqueta2Y = 250
raqueta2DY = 5
puntos1 = 0
puntos2 = 0
tipoLetra = pygame.font.SysFont('arial', 96)
tipoLetra2 = pygame.font.SysFont('arial', 24)
```

```
#-----#
# Creación de la pantalla de juego (SURFACE)
#-----#

visor = pygame.display.set_mode((800,600),FULLSCREEN)

#-----#
# Funciones del programa
#-----#

def pausa():
    # Esta función hace que se espera hasta que se pulse una tecla
    esperar = True
    while esperar:
        for evento in pygame.event.get():
            if evento.type == KEYDOWN:
                esperar = False

def mostrarIntro():
    # Muestra la pantalla de inicio y espera
    mensaje1 = 'P O N G'
    texto1 = tipoLetra.render(mensaje1, True, BLANCO)
    mensaje2 = 'Pulsa una tecla para comenzar'
    texto2 = tipoLetra2.render(mensaje2, True, BLANCO)
    visor.blit(texto1, (50,250,200,100))
    visor.blit(texto2, (400,292,350,30))
    pygame.display.update()
    pausa()

def dibujarJuego():
    # Dibuja la mesa, la pelota, las raquetas y los marcadores
    # Primero borra la pantalla en negro
    visor.fill((0,0,0))
    # Dibuja la pelota y las raquetas
    pygame.draw.circle(visor, BLANCO, (pelotaX,pelotaY),4,0)
    pygame.draw.rect(visor, BLANCO, (raquetaX,raquetaY,10,50))
    pygame.draw.rect(visor, BLANCO, (raqueta2X,raqueta2Y,10,50))
    # Dibuja la red
    for i in range(10):
        pygame.draw.rect(visor, BLANCO, (398,10+60*i,4,30))
    # Dibuja los marcadores
    marcador1 = tipoLetra.render(str(puntos1), True, BLANCO)
    marcador2 = tipoLetra.render(str(puntos2), True, BLANCO)
    visor.blit(marcador1, (300,20,50,50))
    visor.blit(marcador2, (450,20,50,50))
    # Y, finalmente, lo vuelca todo en pantalla
    pygame.display.update()

def decirGanador():
    # Decir qué jugador ha ganado y esperar
```

```

if puntos1 == 11:
    ganador = 'Jugador 1'
else:
    ganador = 'Jugador 2'
mensaje = 'Ganador: ' + ganador
texto = tipoLetra.render(mensaje, True, BLANCO)
visor.blit(texto, (80,350,600,100))
pygame.display.update()
pausa()

#-----#
# Cuerpo principal del juego
#-----#

mostrarIntro()

while True:

    #-----#
    # Gestionar la velocidad del juego
    #-----#

    if pygame.time.get_ticks()-tiempo < 1000/fps:
        continue
    tiempo = pygame.time.get_ticks()

    #-----#
    # Bucle de eventos: Mirar si se quiere terminar el juego
    #-----#

    for evento in pygame.event.get():
        if evento.type == KEYDOWN:
            if evento.key == K_ESCAPE:
                pygame.quit()
                sys.exit()

    #-----#
    # Mover la pelota
    #-----#

    # Primero hay que vigilar por si hay que cambiar de dirección
    # Mira si se impacta con el jugador 1
    diff1 = pelotaY-raquetaY
    if pelotaX == raquetaX + 10 and diff1 >= 0 and diff1 <= 50:
        pelotaDX = -pelotaDX
    # Mira si se impacta con el jugador 2
    diff2 = pelotaY-raqueta2Y
    if pelotaX == raqueta2X and diff2 >= 0 and diff2 <= 50:
        pelotaDX = -pelotaDX
    # Mira si se ha llegado al borde de la pantalla

```

```
if pelotaY < 5 or pelotaY > 595:
    pelotaDY = -pelotaDY
# Mueve la pelota
pelotaX += pelotaDX
pelotaY += pelotaDY

#-----#
# Mover las raquetas
#-----#

# Mira si el jugador 1 mueve la raqueta
teclasPulsadas = pygame.key.get_pressed()
if teclasPulsadas[K_a]:
    raquetaY += raquetaDY
if teclasPulsadas[K_q]:
    raquetaY -= raquetaDY
# Vigilar que la raqueta no se salga de la pantalla
if raquetaY < 0:
    raquetaY = 0
elif raquetaY > 550:
    raquetaY = 550
# Ahora hacemos lo mismo con el jugador 2
if teclasPulsadas[K_l]:
    raqueta2Y += raqueta2DY
if teclasPulsadas[K_p]:
    raqueta2Y -= raqueta2DY
if raqueta2Y < 0:
    raqueta2Y = 0
elif raqueta2Y > 550:
    raqueta2Y = 550

#-----#
# Mirar si se ha ganado un punto
#-----#

# Primero, mira si la pelota ha llegado al borde
if pelotaX > 800 or pelotaX < 0:
    # En tal caso, recolocar juego y cambiar puntuación
    time.sleep(1)
    # Se resitúa las raquetas
    raquetaY = 250
    raqueta2Y = 250
    # Se mira a quien le corresponde el punto
    if pelotaX > 800:
        puntos1 = puntos1 + 1
    else:
        puntos2 = puntos2 + 1
    # Se resitúa a la pelota y se cambia su dirección
    pelotaX = 400
    pelotaDX = -pelotaDX
```

```

# Comprobar si el juego se ha acabado
if puntos1 == 11 or puntos2 == 11:
    decirGanador()
    puntos1 = 0
    puntos2 = 0
    visor.fill((0,0,0))
    mostrarIntro()

#-----#
# Dibujar el juego en pantalla
#-----#

dibujarJuego()

```

Para ver si se ha acabado el juego, debemos ir a la sección en la que se implementa el que se gane un punto y comprobar si una de las dos puntuaciones es ya 11 (para no hacer las partidas muy largas, las jugaremos a 11 puntos):

```

# Comprobar si el juego se ha acabado
if puntos1 == 11 or puntos2 == 11:
    decirGanador()

```

Bien. Dentro del **if** en el que comprobamos que uno de los dos jugadores ha ganado, debemos hacer varias cosas. Lo primero es que el ordenador muestre quien ha ganado. Como esto significa, a su vez, hacer varias cosas más, escribimos el nombre de una función que se encargará de ello, **decirGanador()**, y luego implementaremos su código.

Una vez hecho lo anterior, debemos prepararnos para la posibilidad de una nueva partida. Hay muchas maneras de hacerlo, pero una simple (y que quede bien) es mostrar de nuevo la pantalla de inicio del juego. ¡Qué bien nos viene el haber hecho eso antes como una función! Ahora basta invocarla con su nombre, **mostrarIntro()**. Dicho sea de paso, esta función incorporaba a su vez una pausa, lo que nos viene de perillas.

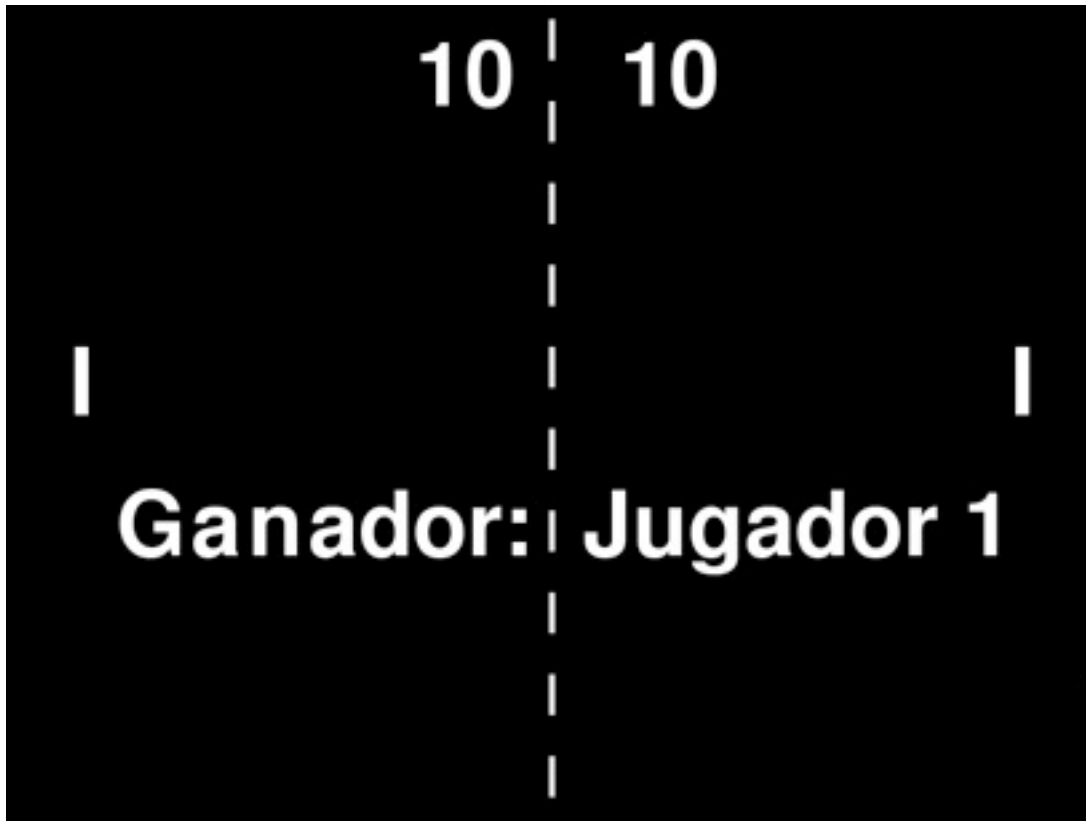
```
mostrarIntro()
```

Dos detalles. El primero es que, por supuesto, para volver a empezar hay que resetear los marcadores.

```
puntos1 = 0
puntos2 = 0
```

Y el segundo, que es fácil de ver si se ejecuta el programa tal cual, es que si no borramos la pantalla previamente, el texto de la pantalla de inicio se superpone a lo que ya está en pantalla, haciéndolo ilegible. Por eso hay que añadir

```
visor.fill((0,0,0))
```

¡Un momento! ¿Te has dado cuenta? Vale, el jugador ha ganado ya que ha alcanzado los puntos necesarios, pero... ¡El marcador no los muestra!. ¿Ves donde está el error?

En efecto, esto ocurre porque hemos puesto la gestión del final de partida antes de que el programa alcance el código de volcar en pantalla la situación del juego. De ello se encarga esa última línea **dibujarJuego()**.

Para corregirlo sólo hemos de cambiar el código anterior que está antes de ella, y ponerlo después.

Solucionado.

Pong.py

En este último paso, vamos a terminar y escribir la versión definitiva del juego. Vamos a añadir los últimos detalles que dejan al juego con un aspecto más profesional; una imagen de fondo, algo de color, sonidos... Éste será el aspecto:



Vayamos con el código final. Como siempre, en negrita, los cambios:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

#-----#
# Pong.py
# Versión definitiva, con sonidos e imagen de inicio
#-----#

#-----#
# Importación de los módulos necesarios
#-----#

import pygame, sys
import time
from pygame.locals import *

#-----#
# Inicialización de Pygame
#-----#

pygame.mixer.pre_init(44100,16,2,1024)
pygame.init()

#-----#
# Definición de variables
#-----#

fps = 60
tiempo = 0
BLANCO = (255,255,255)
AMARILLO = (255,255,0)
pelotaX = 50
pelotaY = 50
pelotaDX = 5
pelotaDY = 5
raquetaX = 50
raquetaY = 250
raquetaDY = 5
raqueta2X = 740
raqueta2Y = 250
raqueta2DY = 5
puntos1 = 0
puntos2 = 0
tipoLetra = pygame.font.Font('Grandezza.ttf', 96)
tipoLetra2 = pygame.font.Font('Grandezza.ttf', 24)
tipoLetra3 = pygame.font.Font('Grandezza.ttf', 48)
sonidoPelota = pygame.mixer.Sound('sonidoPelota.wav')
sonidoRaqueta = pygame.mixer.Sound('sonidoRaqueta.wav')
sonidoError = pygame.mixer.Sound('sonidoError.wav')
```

```
sonidoAplausos = pygame.mixer.Sound('sonidoAplausos.wav')
sonidoLaser = pygame.mixer.Sound('sonidoLaser.wav')
imagenDeFondo = 'pingpong.jpg'

#-----#
# Creación de la pantalla de juego (SURFACE)
#-----#

visor = pygame.display.set_mode((800,600),FULLSCREEN)

#-----#
# Funciones del programa
#-----#

def pausa():
    # Esta función hace que se espera hasta que se pulse una tecla
    esperar = True
    while esperar:
        for evento in pygame.event.get():
            if evento.type == KEYDOWN:
                esperar = False
    sonidoLaser.play()

def mostrarIntro():
    # Muestra la pantalla de inicio y espera
    fondo = pygame.image.load(imagenDeFondo).convert()
    visor.blit(fondo, (0,0))
    mensaje1 = 'PONG'
    texto1 = tipoLetra.render(mensaje1, True, AMARILLO)
    mensaje2 = 'Pulsa una tecla para comenzar'
    texto2 = tipoLetra2.render(mensaje2, True, BLANCO)
    visor.blit(texto1, (50,100,200,100))
    visor.blit(texto2, (235,340,350,30))
    pygame.display.update()
    pausa()

def dibujarJuego():
    # Dibuja la mesa, la pelota, las raquetas y los marcadores
    # Primero borra la pantalla en negro
    visor.fill((0,0,0))
    # Dibuja la pelota y las raquetas
    pygame.draw.circle(visor, BLANCO, (pelotaX,pelotaY),4,0)
    pygame.draw.rect(visor, BLANCO, (raquetaX,raquetaY,10,50))
    pygame.draw.rect(visor, BLANCO, (raqueta2X,raqueta2Y,10,50))
    # Dibuja la red
    for i in range(10):
        pygame.draw.rect(visor, BLANCO, (398,10+60*i,4,30))
    # Dibuja los marcadores
    marcador1 = tipoLetra.render(str(puntos1), True, BLANCO)
    marcador2 = tipoLetra.render(str(puntos2), True, BLANCO)
```

```

visor.blit(marcador1, (300,20,50,50))
visor.blit(marcador2, (450,20,50,50))
# Y, finalmente, lo vuelca todo en pantalla
pygame.display.update()

def decirGanador():
    # Decir qué jugador ha ganado y esperar
    sonidoAplausos.play()
    if puntos1 == 11:
        ganador = 'Jugador 1'
    else:
        ganador = 'Jugador 2'
    mensaje = 'Ganador: ' + ganador
    texto = tipoLetra3.render(mensaje, True, AMARILLO)
    visor.blit(texto, (110,350,600,100))
    pygame.display.update()
    pausa()

#-----#
# Cuerpo principal del juego
#-----#

pygame.mouse.set_visible(False)
mostrarIntro()

time.sleep(0.75)

while True:

    #-----#
    # Gestionar la velocidad del juego
    #-----#

    if pygame.time.get_ticks()-tiempo < 1000/fps:
        continue
    tiempo = pygame.time.get_ticks()

    #-----#
    # Bucle de eventos: Mirar si se quiere terminar el juego
    #-----#

    for evento in pygame.event.get():
        if evento.type == KEYDOWN:
            if evento.key == K_ESCAPE:
                pygame.quit()
                sys.exit()

    #-----#
    # Mover la pelota
    #-----#

```

```
# Primero hay que vigilar por si hay que cambiar de dirección
# Mira si se impacta con el jugador 1
diff1 = pelotaY-raquetaY
if pelotaX == raquetaX + 10 and diff1 >= 0 and diff1 <= 50:
    pelotaDX = -pelotaDX
    sonidoRaqueta.play()
# Mira si se impacta con el jugador 2
diff2 = pelotaY-raqueta2Y
if pelotaX == raqueta2X and diff2 >= 0 and diff2 <= 50:
    pelotaDX = -pelotaDX
    sonidoRaqueta.play()
# Mira si se ha llegado al borde de la pantalla
if pelotaY < 5 or pelotaY > 595:
    pelotaDY = -pelotaDY
    sonidoPelota.play()
# Mueve la pelota
pelotaX += pelotaDX
pelotaY += pelotaDY

#-----#
# Mover las raquetas
#-----#

# Mira si el jugador 1 mueve la raqueta
teclasPulsadas = pygame.key.get_pressed()
if teclasPulsadas[K_a]:
    raquetaY += raquetaDY
if teclasPulsadas[K_q]:
    raquetaY -= raquetaDY
# Vigilar que la raqueta no se salga de la pantalla
if raquetaY < 0:
    raquetaY = 0
elif raquetaY > 550:
    raquetaY = 550
# Ahora hacemos lo mismo con el jugador 2
if teclasPulsadas[K_l]:
    raqueta2Y += raqueta2DY
if teclasPulsadas[K_p]:
    raqueta2Y -= raqueta2DY
if raqueta2Y < 0:
    raqueta2Y = 0
elif raqueta2Y > 550:
    raqueta2Y = 550

#-----#
# Mirar si se ha ganado un punto
#-----#

# Primero, mira si la pelota ha llegado al borde
if pelotaX > 800 or pelotaX < 0:
```

```

# En tal caso, recolocar juego y cambiar puntuación
sonidoError.play()
time.sleep(1)
raquetaY = 250
raqueta2Y = 250
if pelotaX > 800:
    puntos1 = puntos1 + 1
else:
    puntos2 = puntos2 + 1
pelotaX = 400
pelotaDX = -pelotaDX

#-----#
# Dibujar el juego en pantalla
#-----#

dibujarJuego()

#-----#
# Comprobar si el juego se ha acabado
#-----#

if puntos1 == 11 or puntos2 == 11:
    decirGanador()
    puntos1 = 0
    puntos2 = 0
    visor.fill((0,0,0))
    mostrarIntro()

```

Empecemos con los sonidos. ¿Cuáles necesitamos? Lógicamente, uno para cuando la pelota da en una raqueta y otro para cuando aquella rebota en los bordes de la pantalla. Podemos añadir otro para cuando uno de los jugadores falle y le de un punto al otro. Y otro más para las transiciones del juego, cuando el usuario debe pulsar una tecla tras una pausa. No nos podemos olvidar

¿Dónde conseguir esos sonidos? Si sabemos, podemos crearlos nosotros mismos con el programa adecuado (y con una grabadora si la necesitamos). Para quien no sabe hacerlo o para quien quiere reusar el material que otros han hecho y han deseado compartir, en Internet podemos encontrar varios buscadores de sonidos libres. Los que se usan en este programa pueden encontrarse con ayuda de **Soungle** (<http://soungle.com/>). Por claridad, los archivos resultantes se han renombrado. Están en formato **wav**, aunque Pygame soporta también otros, como **mp3** u **ogg vorbis**.

Para incluirlos, el código es el siguiente:

```

sonidoPelota = pygame.mixer.Sound('sonidoPelota.wav')
sonidoRaqueta = pygame.mixer.Sound('sonidoRaqueta.wav')
sonidoError = pygame.mixer.Sound('sonidoError.wav')
sonidoAplausos = pygame.mixer.Sound('sonidoAplausos.wav')
sonidoLaser = pygame.mixer.Sound('sonidoLaser.wav')

```

Como puede verse es muy sencillo. Basta llamar a la función **pygame.mixer.Sound()** y pasarle como argumento el archivo de sonido deseado. El resultado es un objeto de tipo **Sound** que puede usarse ya en el programa. Cada uno de los diferentes sonidos, lo hemos almacenado en una variable para futuras referencias. Los nombres son elocuentes por sí mismos; **sonidoPelota** (cuando rebota en los bordes de la pantalla), **sonidoRaqueta** (cuando una raqueta le da a la pelota), **sonidoError** (cuando se falla y el jugador contrario gana un punto), **sonidoAplausos** (cuando uno de los jugadores gana) y **sonidoLaser** (cuando se pulsa una tecla y se cambia de pantalla).

¿Dónde hacemos que suenen? Las transiciones de pantalla ocurren en dos lugares; después de mostrar la pantalla de inicio cuando va a comenzar el juego y cuando se ha mostrado el ganador de la partida y se va a dar paso, de nuevo, a la pantalla de inicio.

Afortunadamente, como hemos sido ordenados, podemos hacer ambos casos a la vez; siempre ocurre tras pulsar una tecla y eso es en la función **pausa()**, así que basta con que pongamos como última instrucción de dicha función lo siguiente:

```
sonidoLaser.play()
```

En efecto, reproducir un sonido es tan simple como usar el método **play()** que tiene todo objeto o variable de tipo **Sound**.

Los aplausos se han de producir también en un lugar muy concreto; cuando se ha ganado y se muestra al ganador. ¿Adivinas donde ponerlo? Pues sí, al comienzo de la función **decirGanador()**. Y, naturalmente, será así:

```
sonidoAplausos.play()
```

Análogamente, pondremos **sonidoPelota.play()** dentro del **if** que comprueba cuando ésta llega al borde de la pantalla y modifica su dirección. Y **sonidoRaqueta.play()** dentro del **if** que comprueba el impacto entre la pelota y la raqueta. Finalmente, **sonidoError.play()** irá en el interior del **if** que detecta si se ha ganado un punto. ¿No es sencillo una vez que sabes hacerlo?

Pasemos a poner algo de color. Para que no quede todo tan monóculo y aburrido, le daremos un toque de amarillo; pondremos de este color al nombre del programa y al texto que notifica quien ha ganado. Por supuesto, tendremos que definirlo:

```
AMARILLO = (255,255,0)
```

Una vez hecho esto, basta ir a la línea donde se muestra el color deseado y cambiar su **BLANCO** por el nuevo **AMARILLO**.

A pesar de todo, la pantalla de inicio sigue siendo bastante sosa. ¿Por qué no le ponemos una imagen de fondo que quede atractiva? Dicho y hecho.

Podemos proceder de la misma manera que con los sonidos; la hacemos nosotros mismos o buscamos por internet una imagen de licencia libre. De entre los muchos buscadores de imágenes con licencias **Creative Commons**, la imagen usada la hemos encontrado en **flickrCC** (<http://flickrcc.bluemountains.net/index.php>). La hemos redimensionado (con un programa de retoque, **GIMP** por ejemplo) para que tenga el tamaño de la pantalla, la hemos renombrado y la almacenamos en una variable para su uso posterior:

```
imagenDeFondo = 'pingpong.jpg'
```


Recuerda que tanto con los sonidos, como con las imágenes, como con cualquier otro elemento que incorpores al programa, si indicas únicamente el nombre del archivo, tu programa intentará encontrarlo **en la misma carpeta** que en la que se encuentra. Si no está allí, tu juego se cerrará con un error.

¿Dónde cargamos la imagen? Como es evidente, en la función **mostrarIntro()**. Como previamente la hemos almacenado en una variable, sólo queda cargarla en memoria y volcarla en la **SURFACE**:

```
fondo = pygame.image.load(imagenDeFondo).convert()
visor.blit(fondo, (0,0))
```

Ya que estamos en este punto, hay unos cuantos detalles que atar...

Para empezar, queda feo que aquí (y, de hecho, en todo el juego), el puntero del cursor es visible. Lo mejor es volverlo **invisible** y qué mejor momento que justo antes de que el programa invoque a la función **mostrarIntro()**. Así que añadiremos

```
pygame.mouse.set_visible(False)
```

pygame.mouse gestiona, como puede imaginarse, el ratón. Su método **set_visible()** permite mostrar u ocultar el cursor según le pasemos como argumento el valor **True** o **False**. Simple, ¿no?

Otro punto que hay que depurar es que, tras la pantalla de inicio, cuando pulsas una tecla el juego comienza inmediatamente y resulta algo incómodo pues el jugador apenas está preparado. La solución es sencilla, añadir un pequeño retardo con

```
time.sleep(0.75)
```

justo detrás de la llamada a **mostrarIntro()**.

El detalle final tiene que ver con varios factores distintos. Habrás notado que, al cambiar la pantalla de inicio y poner una imagen de fondo, el texto que poníamos en pantalla no queda bien situado. Por otra parte, y esto es más grave, si el ordenador en el que pruebas el juego no tiene el tipo de letra **arial** instalado (o lo tiene de forma defectuosa o es diferente) el aspecto del texto puede ser muy distinto. Por poner un ejemplo, los caracteres pueden quedar muy grandes y no caber en pantalla...

Es mejor no dejar nada al azar: Lo que suele hacerse es incorporar con el programa un tipo de letra específico de manera que, sea cual sea el ordenador y la plataforma en la que se ejecute, el aspecto sea siempre el mismo. De nuevo, usamos un buscador de **tipos de letra libres** como **Simply the Best** (<http://simplythebest.net/fonts/>).

ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
1234567890

En el programa usamos el tipo **Grandezza**, de formas curvas y elegantes. Si eliges un formato **TrueType (ttf)** te valdrá para cualquier plataforma y tamaño (pues son vectoriales). Renombrado y colocado en la carpeta del juego, cargamos el archivo que contiene el tipo de letra así:

```
tipoLetra = pygame.font.Font('Grandezza.ttf', 96)
tipoLetra2 = pygame.font.Font('Grandezza.ttf', 24)
tipoLetra3 = pygame.font.Font('Grandezza.ttf', 48)
```

Fíjate en la diferencia. En las versiones anteriores del juego usábamos la función de Pygame **pygame.font.SysFont()**. Ahora usamos **pygame.font.Font()** y le pasamos dos argumentos; el nombre del archivo con el tipo de letra y el tamaño que deseamos.

Hemos elegido tres tamaños diferentes. Uno muy grande (**tipoLetra**, de 96 puntos) para el título del juego, otro pequeño (**tipoLetra2**, de 24 puntos) para el mensaje de pulsar una tecla y otro intermedio (**tipoLetra3**, de 48 puntos) para notificar quién es el ganador del juego. Sólo queda resituar los textos y adjudicarles los nuevos tipos. Las modificaciones, allí donde estaban, son las siguientes:

```
mensaje1 = 'PONG'

visor.blit(texto1, (50,100,200,100))
visor.blit(texto2, (235,340,350,30))
```

(se quitan los espacios entre las letras de PONG para que quede mejor y se colocan los dos textos de **mostrarIntro()** en nuevas posiciones)

```
texto = tipoLetra3.render(mensaje, True, AMARILLO)
visor.blit(texto, (110,350,600,100))
```

(se redimensiona el texto de felicitación en **decirGanador()** a **tipoLetra3** y se cambia su posición para que quede centrado)

¿Hemos acabado? ¡No, aún hay algo más! Según como sea tu ordenador o como tengas configurado tu sistema, es muy posible que el sonido se escuche retardado y no haya sincronización, por ejemplo, entre el golpe de raqueta y pelota y el ruido de rebote. Hay varias causas posibles, pero lo más sencillo es **inicializar por nuestra cuenta el sistema de sonido** y no dejárselo de forma automática a Pygame. Añadiremos para ello, antes de **pygame.init()** la siguiente línea:

```
pygame.mixer.pre_init(44100,16,2,1024)
```

Existe, en realidad, dos funciones diferentes para hacer esta tarea pero la que usamos es la más habitual cuando se inicializa de forma global a Pygame. **pygame.mixer** es el módulo que se encarga de gestionar música, sonido y vídeo; su método **pre_init()** indica a Pygame que cuando sea inicializado use los parámetros indicados. Los valores concretos de éstos dependen del tipo de sonido o música que usemos (sus características las podemos ver en el mismo lugar del que nos los hemos descargado o con un programa como de edición de audio como **Audacity**).

Por si te interesa profundizar algo más, los dos primeros valores son la **frecuencia de muestreo** del sonido y cuántos bits se usarán para cada **muestra** de audio. El siguiente

parámetro es para indicar si queremos que se reproduzca en **mono** (1) o en **estéreo** (2). El último es lo que se conoce como tamaño del **buffer**. Básicamente es el número de muestras internas que se usarán en el mezclador de sonido. Un valor bajo puede ser insuficiente para realizar el trabajo, pero uno grande puede ralentizar el proceso.

Para el tipo de sonidos que hemos usado en nuestro juego, los valores anteriores funcionan correctamente.

Es muy posible que esa línea no te haga falta en caso de que tu sistema operativo sea **MacOS X**. ¿Cómo hacer que funcione? Python tiene una función que permite determinar cual es el sistema operativo en el que se está ejecutando. Como puede imaginarse, dicha función se encuentra en el módulo **sys** y se llama **sys.platform()**. Si usas **Windows**, devuelve la cadena de texto **'win32'** (si tu sistema es de 32 bits, pero en cualquier caso empieza por **'win'**), si usas **MacOS X** el texto es **'darwin'** y en el caso de un Linux como **Ubuntu**, **'linux2'** (comenzando por **'linux'**). Siempre puedes usar esta función al principio de tu programa, comprobarla con un **if** y, según cual sea el resultado, cargar o no la línea de **pygame.mixer**.

¡Finalizado!

Como puedes ver, el resultado final es bastante aparente.

Bien, esta aquí hemos llegado. Pueden mejorarse muchas cosas y añadir características como música, efectos especiales y muchas más.

¿Te animas?...