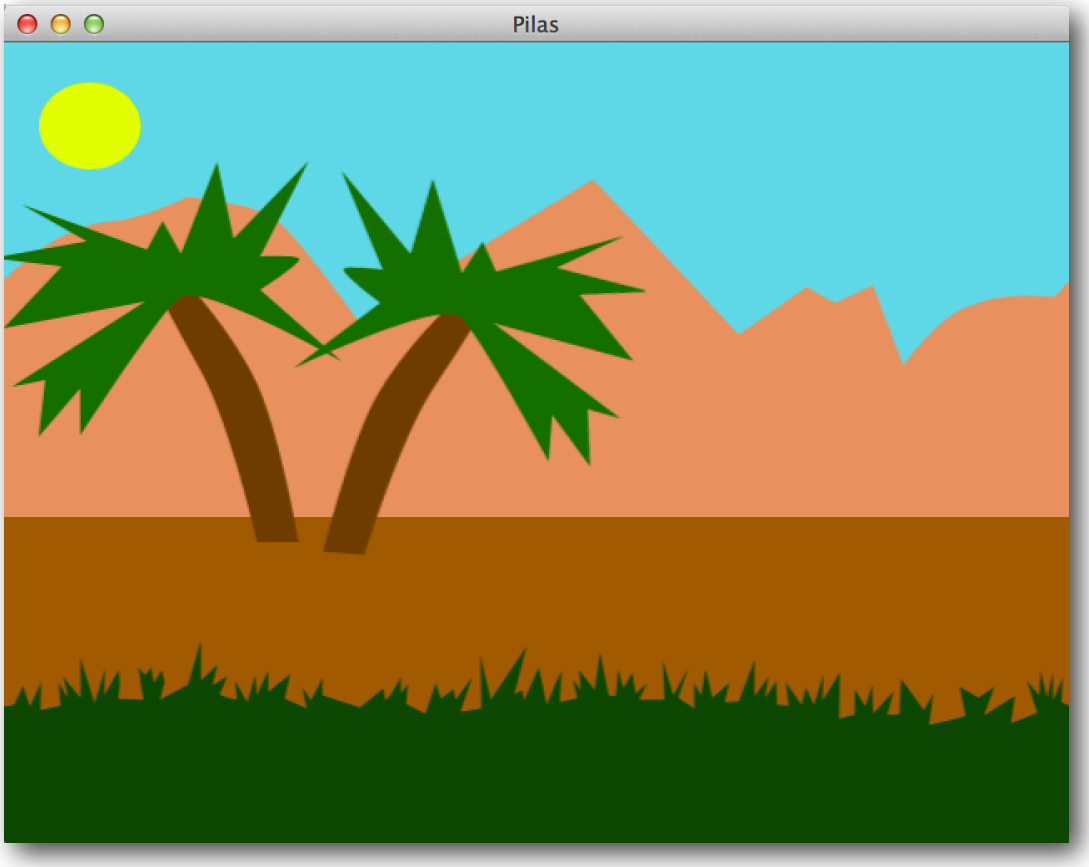
Scrolling

Un nuevo paso podemos darlo observando la técnica en el caso de esos juegos en los que los fondos se desplazan a medida que el jugador avanza. Una muestra la puedes ver dentro del propio Pilas, en el ejemplo denominado **Fondos**:



Allí puedes estudiar el uso de la coordenada **z** de los actores y del tipo de fondo específico **pilas.fondos.Desplazamiento** para conseguir varios planos que se mueven a diferentes velocidades, algo que es muy vistoso.

Pero nosotros vamos a ir por otros derroteros. Para empezar vamos a tener un fondo compuesto por varias imágenes y vamos a ir desplazándolo verticalmente. Y vamos a ver cómo conjugamos ello con un actor que se nos mueva por allí y tenga alguna interacción. Por el camino, manejaremos el concepto de camara, usaremos las ya nombradas coordenadas **z** y veremos otros puntos de interés.

¿Te animas?

desplazando01.py

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-
#---------------------------------------------------------------------
# desplazando01.py
# Movimiento de fondos
#---------------------------------------------------------------------

import pilas

pilas.iniciar(ancho=500, alto=500)

paredes1 = pilas.actores.Actor('arbustos.png')
paredes2 = pilas.actores.Actor('ladrillos.png', y=1000)
paredes3 = pilas.actores.Actor('maderas.png', y=2000)

nubes = pilas.fondos.Fondo('nubes.jpg')
nubes.fijo = True

pilas.escena_actual().camara.y = [2250], 20

pilas.ejecutar()
```

En lo primero en lo que tienes que fijar tu atención es en la línea

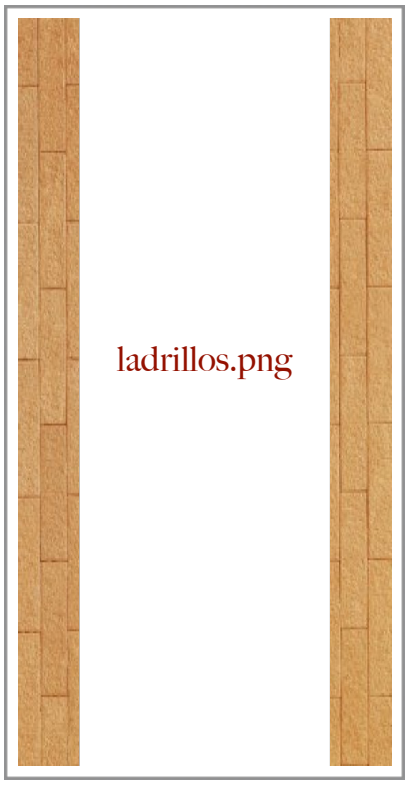
```
pilas.escena_actual().camara.y = [2250], 20
```

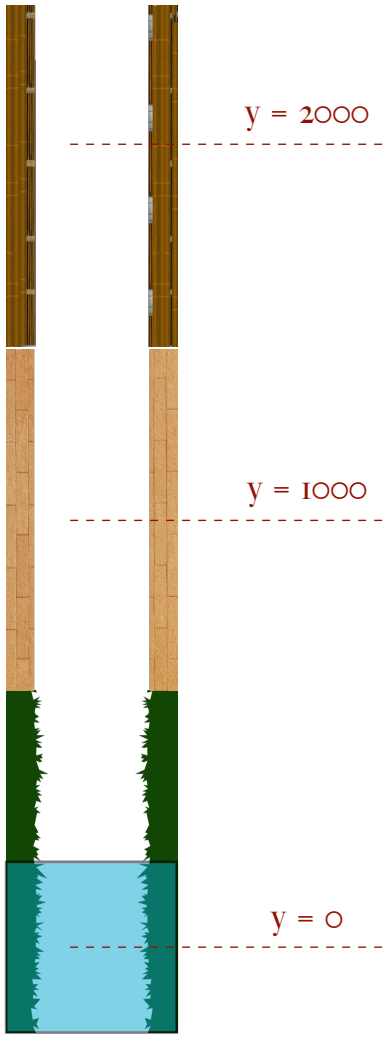
Bien. En todos los tutoriales anteriores el centro de la ventana de Pilas se correspondía con el origen de coordenadas, **x=0** e **y=0**. Pero esto no tiene que ser necesariamente así. Como si de una película se tratara, en realidad estamos mirando la escena que estamos creando a través de una cámara y, sí, por defecto apunta al origen de coordenadas (si has trabajado con programas de modelado 3D, como **Blender**, te resultará familiar). La cámara, de hecho, es un objeto también y podemos controlar sus coordenadas x e y, consiguiendo de esta manera que nuestra visión se desplace por diferentes puntos de la escena.

¿Cómo accedemos a la cámara? Con su dependencia con el concepto de escena (el cual veremos con más detalle en el proximo tutorial), es fácil de imaginar que se trata del lógico **pilas.escena_actual().camara**.

Recuerda, también, que si a un atributo de posición le pasamos una lista, Pilas produce una animación de movimiento con esos elemetos. Y si, además, le pasamos otro número, Pilas lo usa como el tiempo que debe emplear para hacerlo. En este caso, **20** segundos.

Viendo las imágenes que hemos usado, es posible que lo comprendas mejor:





Se trata de imágenes transparentes de **500**x**1000** pixeles de tamaño. La primera de ellas, **arbustos.png**, la creamos en su posición por defecto (es decir, **x=0** e **y=0**), centrada en la ventana de Pilas y la cámara (indicadas en azul; de paso observa que hemos declarado la ventana de Pilas de **500**x**500** píxeles). La segunda la creamos encima, es decir, su centro tiene que tener de coordenada **y=1000**, ya que tienen **1000** píxeles de alto. ¿Lo ves en la definición del actor correspondiente a la imagen **ladrillos.png**? De la misma manera, otros **1000** píxeles más arriba (**y=2000**) creamos el actor de **maderas.png**.

¿Cómo debemos mover la cámara durante esos **20** segundos? No hemos de cambiar su coordenada **x**, pues queremos que simplemente suba. ¿Hasta qué valor de **y**? Si usamos **y=2000** quedará centrada en mitad de la imagen de la madera, que tiene **1000** pixeles de alto; quedarán por encima y por debajo un total de **500** pixeles. Ahí está la clave; como queda la mitad, **250** píxeles, sin cubrir por encima, eso es lo que hemos de subir de más a la cámara. De ahí la cifra que hemos usado de **2250** para su movimiento.

El último detalle es el uso de otra imagen de fondo para conseguir una sensación de mayor realismo en el movimiento.

El truco es no sólo ponerla como fondo, si no usar el atributo **fijo**,

```
nubes = pilas.fondos.Fondo('nubes.jpg')
nubes.fijo = True
```

lo que hace que un objeto de Pilas permanezca fijo con respecto a la cámara, siguiéndola, y no se desplace de forma aparentemente.

Guarda el código con el nombre **desplazando01.py** y ejecútalo.

Queda bien, ¿eh?

desplazando02.py

```
#! /usr/bin/env python
# -*- coding:utf-8 -*-
#-----------------------------------------------------------------------
# desplazando02.py
# Añadiendo un actor que salta y rebota
#-----------------------------------------------------------------------

import pilas

pilas.iniciar(ancho=500, alto=500)

# Definimos las teclas que moverán al personaje
teclas = {pilas.simbolos.IZQUIERDA: 'izquierda',
          pilas.simbolos.DERECHA: 'derecha'}

# Creamos un control personalizado con esas teclas
mandos = pilas.control.Control(pilas.escena_actual(), teclas)


# Definimos la clase de actor
class Bicho(pilas.actores.Actor):
    "Un actor que se mueve con las flechas del cursor y con animación"

    def __init__(self):
        pilas.actores.Actor.__init__(self)
        self.imagen = pilas.imagenes.cargar_imagen('renata.png')
        # Hacemos que el actor se mueva con el comportamiento personalizado
        self.hacer(Esperando())


# Definimos el comportamiento es espera
class Esperando(pilas.comportamientos.Comportamiento):
    "Actor en posicion normal o esperando a que el usuario pulse alguna tecla"

    def iniciar(self, receptor):
        self.receptor = receptor

    def actualizar(self):
        if mandos.izquierda or mandos.derecha:
            self.receptor.hacer(Saltando())


# Definimos el comportamiento de salto
class Saltando(pilas.comportamientos.Comportamiento):
    "Actor Saltando"
```

```
    def iniciar(self, receptor):
        self.receptor = receptor
        self.origen = self.receptor.y
        self.dy = 10
        if mandos.izquierda:
            self.dx = -4
        elif mandos.derecha:
            self.dx = 4

    def actualizar(self):
        self.receptor.y += self.dy
        self.receptor.x += self.dx
        self.dy -= 0.2

        if abs(self.receptor.x) > 180:
            self.dx = -self.dx

        if self.receptor.y < self.origen:
            self.receptor.y = self.origen
            self.receptor.hacer(Esperando())

# Creación de paredes
paredes1 = pilas.actores.Actor('arbustos.png')
paredes2 = pilas.actores.Actor('ladrillos.png', y=1000)
paredes3 = pilas.actores.Actor('maderas.png', y=2000)

# Creación del fondo fijo
nubes = pilas.fondos.Fondo('nubes.jpg')
nubes.fijo = True

# Creación del protagonista
renata = Bicho()

# Activación del movimiento de la cámara
pilas.escena_actual().camara.y = [2250], 20

pilas.ejecutar()
```

En primer lugar te hemos de presentar a nuestra protagonista, **renata**:

Se trata de una arañita que salta entre paredes y que trata de no caerse. De momento, en este paso, hemos creado la clase actor correspondiente, **Bicho**, y le hemos dotado de un

comportamiento específico en el salto. ¡No te alarmes! Si has seguido el tutorial anterior, esto debería ser fácil para ti...

Observa el código. Verás que las primeras líneas que hemos añadido son conocidas: la definición de las teclas que se van a usar para mover a **renata** (sólo dos; izquierda y derecha para saltar en esas direcciones) y el control personalizdo correspondiente.

A continuación, en la definición de la clase **Bicho**, sólo le hemos asignado su imagen y el comportamiento por defecto, nuestro viejo conocido **Esperando**, a través del método **hacer()** que vimos en el tutorial anterior.

Pasamos al comportamiento clave, **Saltando**, que se nos plantea cuando se ha pulsado la flecha izquierda del cursor o la derecha. ¿Qué es lo que queremos hacer? Nuestra arañita tiene que rebotar en las paredes, así que sólo debemos adaptar ligeramente el comportamiento de salto que definimos para **chuck** (nuevamente, en el tutorial anterior). Por ello, hemos cambiado el control de la velocidad horizontal, que antes era una constante global, **VELOCIDAD**, y ahora es un atributo del comportamiento, **self.dx**. Es importante notar que **Saltando** se ejecuta cuando se pulsa cualquiera de las dos teclas de movimiento, con lo que hemos de darle un signo distinto, según cuál sea la dirección del movimiento que queremos (y que el jugador no pueda modificarlo en mitad del salto). Éste es el método **iniciar()**:

```
def iniciar(self, receptor):
    self.receptor = receptor
    self.origen = self.receptor.y
    self.dy = 10
    if mandos.izquierda:
        self.dx = -4
    elif mandos.derecha:
        self.dx = 4
```

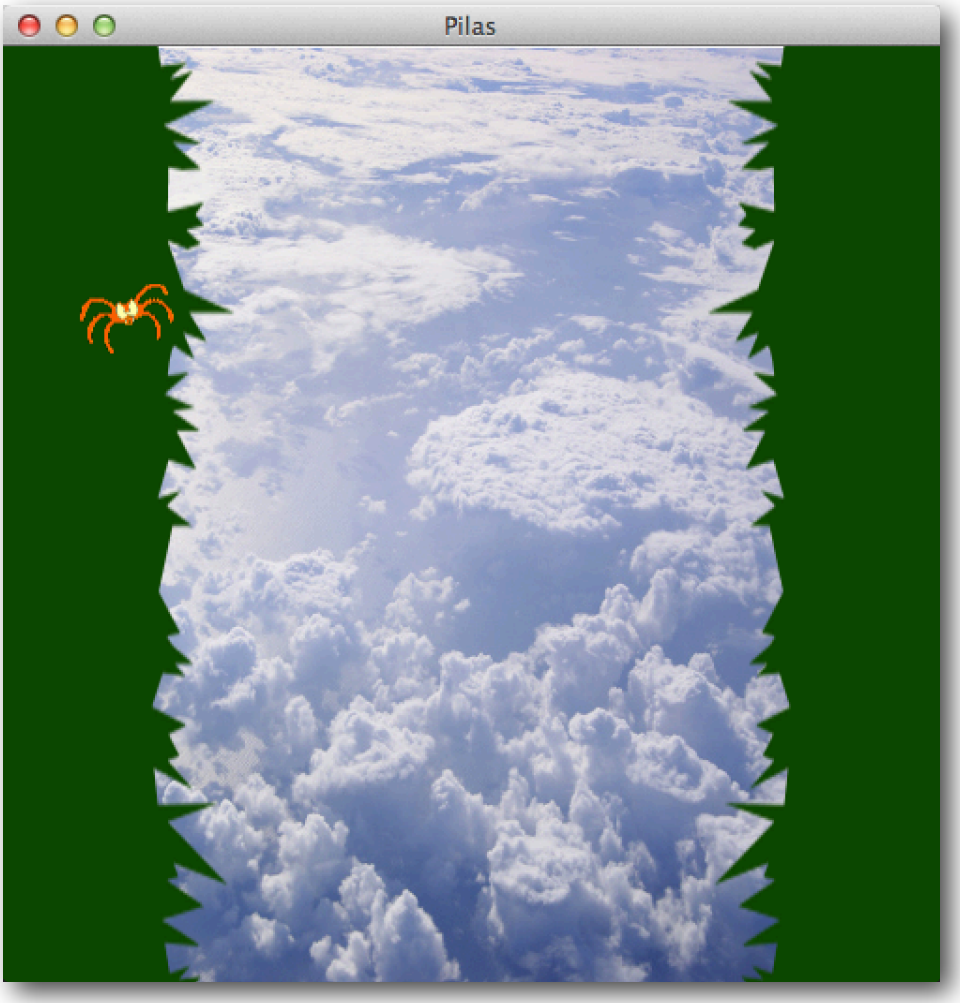
Bien, ya tenemos el movimiento iniciado (con los valores ajustados para que no salte ni mucho ni poco). Queda controlar cuándo acabar el salto (algo que ya vimos) y cuándo rebotar en las paredes. Eso lo encontramos en el método **actualizar()**:

```
def actualizar(self):
    self.receptor.y += self.dy
    self.receptor.x += self.dx
    self.dy -= 0.2

    if abs(self.receptor.x) > 200:
        self.dx = -self.dx

    if self.receptor.y < self.origen:
        self.receptor.y = self.origen
        self.receptor.hacer(Esperando())
```

Si no ocurre lo contrario, el salto finalizará de nuevo cuando se alcance la misma posición vertical que la de partida, algo que no se notará, de todas formas, al ir subiendo la cámara.... Respecto del rebote en las paredes, lo podemos implementar (entre otras formas) mirando cuánto se acerca a los bordes de la ventana. Recordemos que mide **500**

píxeles de ancho y que, por lo tanto, se encuentran en las coordenadas **x=-250** y **x=250**. Dando un cierto margen, de acuerdo al dibujo, podemos determinar el rebote cuando lleguemos a los valores **-200** o **200**. Y podemos hacerlo de un tirón mirando su valor absoluto con la función **abs()**. ¿Ves el **if** en el que se cambia el valor de **self.dx**?

Ya está todo hecho. El resto es un poco de maquillaje; es una buena costumbre añadir comentarios en el código a medida que éste crece...



Venga, ejecuta el código anterior con el nombre **desplazando02.py**, . ¡Qué simpática es **renata**! Pero apenas da tiempo a verla... Para que sea jugable, tendremos que pasar al siguiente apartado.

desplazando03.py

En éste último paso, vamos a dejarlo casi listo para que lo retoques y se convierta en algo jugable. ¿Te animas?

```
#! /usr/bin/env python
# -*- coding:utf-8 -*-
#---------------------------------------------------------------------
# desplazando03.py
# Haciendo que sea jugable
#---------------------------------------------------------------------

import pilas
import random

pilas.iniciar(ancho=500, alto=500)

redes = pilas.grupo.Grupo()

# Definimos las teclas que moverán al personaje
teclas = {pilas.simbolos.IZQUIERDA: 'izquierda',
          pilas.simbolos.DERECHA: 'derecha'}

# Creamos un control personalizado con esas teclas
mandos = pilas.control.Control(pilas.escena_actual(), teclas)


# Definimos la clase de actor
class Bicho(pilas.actores.Actor):
    "Un actor que se mueve con las flechas del cursor y con animación"

    def __init__(self):
        pilas.actores.Actor.__init__(self)
        self.imagen = pilas.imagenes.cargar_imagen('renata.png')
        self.x = -200
        self.texto = None
        # Hacemos que el actor se mueva con el comportamiento personalizado
        self.hacer(Esperando())

    def actualizar(self):
        if self.texto is None:
            if self.y < pilas.escena_actual().camara.y - 250:
                self.texto = pilas.actores.Texto(u'Ohhh... ¡Has Perdido!')
                self.texto.fijo = True
            elif self.y > 2500:
```

```
            self.texto = pilas.actores.Texto(u'¡Muy Bien! ¡Has Ganado!')
            self.texto.fijo = True


# Definimos el comportamiento es espera
class Esperando(pilas.comportamientos.Comportamiento):
    "Actor en posicion normal o esperando a que el usuario pulse alguna tecla"

    def iniciar(self, receptor):
        self.receptor = receptor

    def actualizar(self):
        if mandos.izquierda or mandos.derecha:
            if abs(self.receptor.x) >= 180:
                self.receptor.hacer(Saltando())


# Definimos el comportamiento de salto
class Saltando(pilas.comportamientos.Comportamiento):
    "Actor Saltando"

    def iniciar(self, receptor):
        self.receptor = receptor
        self.origen = self.receptor.y
        self.dy = 12
        if mandos.izquierda:
            self.dx = -4
        elif mandos.derecha:
            self.dx = 4

    def actualizar(self):
        self.receptor.y += self.dy
        self.receptor.x += self.dx
        self.dy -= 0.2

        if self.receptor.x < -200:
            # Saltar hacia el otro lado si procede
            if mandos.derecha:
                self.receptor.hacer(Saltando())

            self.dx = -self.dx

        elif self.receptor.x > 200:
            # Saltar hacia el otro lado si procede
            if mandos.izquierda:
                self.receptor.hacer(Saltando())

            self.dx = -self.dx
```

```
        if self.receptor.y < self.origen:
            self.receptor.y = self.origen
            self.receptor.hacer(Esperando())


# Definimos la clase de actor
class Red(pilas.actores.Actor):
    "La Telaraña que aparece en los laterales"

    def __init__(self):
        pilas.actores.Actor.__init__(self)
        self.imagen = pilas.imagenes.cargar_imagen('red.png')
        # Crear aleatoriamente en un lado o en el otro
        if random.randint(0, 1) == 1:
            self.x = -210
        else:
            self.x = 210
        self.y = pilas.escena_actual().camara.y + 300


# Definimos la función que crea las redes
def crear_red():
    redes.append(Red())
    return True


# Definimos la función que pega la araña a la red
def mantener(ella, red):
    ella.x = red.x
    ella.y = red.y
    red.eliminar()
    ella.hacer(Esperando())

# Creación de paredes
paredes1 = pilas.actores.Actor('arbustos.png')
paredes2 = pilas.actores.Actor('ladrillos.png', y=1000)
paredes3 = pilas.actores.Actor('maderas.png', y=2000)

# Creación del fondo fijo
nubes = pilas.fondos.Fondo('nubes.jpg')
nubes.fijo = True

# Creación del protagonista
renata = Bicho()
```

```
# Creación aleatoria de redes
pilas.mundo.agregar_tarea(2, crear_red)

# Agregar la colisión entre la red y renata
pilas.escena_actual().colisiones.agregar(renata, redes, mantener)

# Activación del movimiento de la cámara
pilas.escena_actual().camara.y = [2250], 20

pilas.ejecutar()
```

¿Cómo hacemos que se juegue? Si permitimos que, justo en los laterales y sólo en ellos, **renata** pueda saltar, podremos conseguir que vaya subiendo. Humm.... Entonces, ¿cuándo se gana? ¿Cuándo se pierde? La victoria será cuando nuestra arañita llegue arriba del todo. Y la derrota cuando se quede más abajo de la cámara. Una forma rápida de indicar esto es en el propio método **actualizar()** de la clase **Bicho**. De paso, hacemos que renata comience desde uno de los laterales (**x=200**) escribiéndolo en su método **__init__()** y allí mismo dejamos preparado el futuro texto que indicará la victoria o la derrota. Así es como nos quedará el **__init__()** de la clase Bicho:

```
def __init__(self):
    pilas.actores.Actor.__init__(self)
    self.imagen = pilas.imagenes.cargar_imagen('renata.png')
    self.x = -200
    self.texto = None
    # Hacemos que el actor se mueva con el comportamiento personalizado
    self.hacer(Esperando())
```

Y así el método **actualizar()**:

```
def actualizar(self):
    if self.texto is None:
        if self.y < pilas.escena_actual().camara.y - 250:
            self.texto = pilas.actores.Texto(u'Ohhh... ¡Has Perdido!')
            self.texto.fijo = True
        elif self.y > 2500:
            self.texto = pilas.actores.Texto(u'¡Muy Bien! ¡Has Ganado!')
            self.texto.fijo = True
```

Aquí puedes ver por qué usamos **None**. Si simplemente creamos un actor de tipo **Texto** cuando **renata** se sale de la pantalla, esto se estará haciendo una y otra vez desde el momento que haya ocurrido. Para hacerlo una única vez, la declaramos como **None** en el **__init__()** y miramos si sigue así cuando proceda; de esta manera, en las sucesivas pasadas no se volverá a crear, al dejar de serlo. Observa las coordenadas para la derrota o

la victoria. Si recuerdas el esquema gráfico que hemos usado al principio, verás que la última imagen, **maderas.png**, llega hasta **y=2500**. Del mismo modo, nuestra ventana de Pilas tiene una altura de **500** píxeles y la cámara está centrada, de manera que si la araña queda **250** píxeles por debajo de la cámara, se saldrá de la pantalla.

Vamos al salto desde los laterales. Si estamos atentos, podemos hacer saltar hacia la derecha a **renata** un instante cuando está en la pared izquierda, y viceversa, como si se apoyara en ésta. Además, hemos de asegurarnos que esto ocurre en los dos modos, **Esperando** y **Saltando**. En el primero vamos a ser menos restrictivos, teniendo en cuenta que la araña puede estar cayendo justo cerca del lateral. Por ello, su método **actualizar()** ha cambiado a

```
def actualizar(self):
    if mandos.izquierda or mandos.derecha:
        if abs(self.receptor.x) >= 180:
            self.receptor.hacer(Saltando())
```

es dcir, en modo **Esperando**, podemos saltar si estamos a menos de **250-180=70** píxeles del borde de la ventana.

¿Y en modo **Saltando**? Veamos cómo ha cambiado su método:

```
def actualizar(self):
    self.receptor.y += self.dy
    self.receptor.x += self.dx
    self.dy -= 0.2

    if self.receptor.x < -200:
        # Saltar hacia el otro lado si procede
        if mandos.derecha:
            self.receptor.hacer(Saltando())

        self.dx = -self.dx

    elif self.receptor.x > 200:
        # Saltar hacia el otro lado si procede
        if mandos.izquierda:
            self.receptor.hacer(Saltando())

        self.dx = -self.dx

    if self.receptor.y < self.origen:
        self.receptor.y = self.origen
        self.receptor.hacer(Esperando())
```

Fíjate en los cambios, que se refieren a la coordenada **x** de **renata** (en el lenguaje del **Comportamiento**, **receptor.x**): Si estamos en una pared, a menos de **250-200=50** píxeles del borde y si pulsamos saltar hacia el otro lado, se salta. Y sólo se salta bajo esas condiciones. En caso contrario, simplemente se rebota, cambiando el sigo de **self.dx**.

(Hay otro detalle, esta vez en el método **iniciar()** y es que, para que sea jugable, **renata** tiene que saltar con algo más de fuerza, así que hemos cambiado su **self.dy** a **12**)

Como última ayuda al jugador, hemos añadido un componente más al juego. Se trata de una telaraña, una red que va apareciendo al azar por las paredes y que permite que, cuando renata la alcanza, pueda descansar esperando y tener un respiro para saltar con más calma.

El tipo de actor es simple, **Red**. En su definición, usamos el módulo **random** (de ahí el **import** adicional al principio del programa) para que se cree en un pared o en la otra, de forma aleatoria. Y hay que ajustar la posición vertical donde va a aparecer el actor que, como es fácil de imaginar, dependerá la cámara:

```
        self.y = pilas.escena_actual().camara.y + 300
```

¿Ves el porqué de **300**? El final de la ventana está **250** píxeles más arriba que la coordenada **y** de la cámara. Y la imagen de la telaraña mide **100** píxeles, así que si su centro está **50** píxeles (la mitad) más arriba al crearse, irá apareciendo por la parte superior de la pantalla a medida que la cámara ascienda.

Ya tenemos la definición de las redes. Ahora queremos que aparezcan digamos cada **2** segundos. ¡Eso ya lo sabemos hacer!:

```
pilas.mundo.agregar_tarea(2, crear_red)
```

En efecto, cada 2 segundos lanzamos la función **crear_red()**:

```
def crear_red():
    redes.append(Red())
    return True
```

redes es un **pilas.grupo.Grupo**, un grupo de actores (como ya indicamos en un tutorial anterior). ¿Ves su definición al principio del código? Así que, en la función **crear_red()** simplemente se crea una red nueva y se añade al grupo. ¡Recuerda el detalle de devolver **True** para que siga realizándose la tarea y no la termine!

Solo queda implementar el que **renata** se pueda apoyar en la red cuando la toca. Eso es una colisión. Y también sabemos agregar colisiones...

```
pilas.escena_actual().colisiones.agregar(renata, redes, mantener)
```

Bien. Cuando **renata** choque con un actor del grupo **redes**, ejecutará la función **mantener()**.

¿Qué es lo que queremos que ocurra? Queremos que deje el salto y descanse, así que hemos de pasar al comportamiento **Esperando**. Además, centraremos su posición con la

que tiene la red y, para evitar problemas de colisiones posteriores de manera fácil (aunque algo grosera), eliminamos la red:

```
def mantener(ella, red):
    ella.x = red.x
    ella.y = red.y
    red.eliminar()
    ella.hacer(Esperando())
```

Recuerda que los nombres de los argumentos son mudos; puedes poner cualquiera y referirte a ellos con el que hayas empleado. En nuestro caso, **ella** se corresponderá con **renata** y **red** con la telaraña del grupo **redes** con la que ha colisionado.

¡Eso es todo! Guarda todo el código anterior con el nombre **desplazando03.py** y ejecútalo. ¡Salta, salta, arañita!

Quedan muchos detalles que depurar para que quede algo decente... Por ejemplo, hay un **bug** en el que **renata** se queda quieta en mitad de las nubes. ¿Lo corregirías? O puedes mejorar la forma en la que se termina el juego, o poner un marcador que mida la altura de la araña, o las redes en las que se apoya. ¿Y por qué han de desaparecer? Y...

¡Como siempre, un mundo de posibilidades!