

# Pilas (Práctica 3)

## Objetivo

Ahora vas a trabajar progresivamente en varios programas hasta conseguir dominar el uso de diferentes elementos, sobre todo, entender cómo se trabaja con **clases** y con **objetos**. Para ello vamos a modificar ligeramente el comportamiento de los programas y vamos a dar una explicación más detallada de cada paso.

## Programa 3: pilas3.py

En **Pilas**, el manejo del bucle de animación es automático y nos tenemos que centrar, en su lugar, en definir el comportamiento de cada elemento. Observa el siguiente código (y lee los comentarios para entender el funcionamiento); cuando se ejecuta el programa, con cada click del ratón se dibuja un cuadrado de un color aleatorio...

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

import pilas
from random import randint # Vamos a necesitar generar números al azar

pilas.iniciar()

# Indicamos que el fondo de la ventana será negro
pilas.fondos.Color(pilas.colores.negro)

# Vamos a crear una clase de objetos que represente a nuestros 'puntos'
# Estos objetos deben ser actores y, por tanto, la clase es derivada de ella.
class Punto(pilas.actores.Actor):
    # El método __init__() debe incluir la posición del punto
    def __init__(self, x=0, y=0):
        # Siempre hay que llamar al __init__() de la clase 'Padre' para
        # que la 'hija' herede sus características.
        # En nuestro caso, como va a ser un actor a partir de una imagen
        # cuadrada que representa el punto, primero generamos la imagen...
        imagen = pilas.imagenes.cargar_superficie(4, 4)
        # ... generamos el color que va a tener el cuadrado al azar...
        color = pilas.colores.Color(randint(0, 255), randint(0, 255),
                                     randint(0, 255))
        # ... y pintamos el cuadrado con dicho color
        imagen.pintar(color)
        # Ahora sí, llamamos al __init__() del padre con la imagen hecha...
        pilas.actores.Actor.__init__(self, imagen)
        # ... ¡y que no se nos olvide! Guardamos la información de la posición
        # del punto, al crearlo, en los correspondientes atributos.
```

```
self.x = x
self.y = y

# Ahora solo queda definir la función que se encargará de crear el punto en
# pantalla cuando hagamos click con el ratón.
def crear_punto(evento):
    # creamos un nuevo punto en la posición en la que se ha producido el evento
    Punto(evento.x, evento.y)

# Conectamos la función con el evento de hacer click con el ratón
pilas.eventos.click_de_mouse.conectar(crear_punto)

# ¡Ya está todo definido! Podemos lanzar el programa...
pilas.ejecutar()
```

## Programa 4: cuadradosDiagonales.py

Bien. Modifiquemos el programa anterior para que, durante la animación, los cuadrados se muevan en diagonal. Las variaciones con respecto al programa anterior van acompañadas con el correspondiente comentario en el código:

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

import pilas
from random import randint

pilas.iniciar()

pilas.fondos.Color(pilas.colores.negro)

class Punto(pilas.actores.Actor):

    def __init__(self, x=0, y=0):
        imagen = pilas.imagenes.cargar_superficie(4, 4)
        color = pilas.colores.Color(randint(0, 255), randint(0, 255),
                                     randint(0, 255))
        imagen.pintar(color)
        pilas.actores.Actor.__init__(self, imagen)
        self.x = x
        self.y = y
```

```
# El método actualizar() nos permite definir comportamientos de los actores
# en cada fotograma de la animación. En este caso, aumentamos las coordena-
# das del punto, provocando que se produzca un movimiento en diagonal.
def actualizar(self):
    self.x += 1
    self.y += 1

def crear_punto(evento):
    Punto(evento.x, evento.y)

pilas.eventos.click_de_mouse.conectar(crear_punto)

pilas.ejecutar()
```

## Programa 5: cuadradosRebotantes.py

Seguimos modificando. Ahora queremos que los cuadrados no se salgan de la ventana y que reboten con los bordes...

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

import pilas
from random import randint

pilas.iniciar()

pilas.fondos.Color(pilas.colores.negro)

class Punto(pilas.actores.Actor):

    def __init__(self, x=0, y=0):
        imagen = pilas.imagenes.cargar_superficie(4, 4)
        color = pilas.colores.Color(randint(0, 255), randint(0, 255),
                                     randint(0, 255))
        imagen.pintar(color)
        pilas.actores.Actor.__init__(self, imagen)
        self.x = x
        self.y = y

# Para poder hacer el rebote al llegar a los extremos de la ventana,
# hemos de poder cambiar la dirección del movimiento. Para ello
```

```
# definimos los atributos dx y dy con el valor inicial de 1.
self.dx = 1
self.dy = 1

def actualizar(self):
    # Ahora, para crear el movimiento, sumamos a la posición la velocidad
    self.x += self.dx
    self.y += self.dy
    # Bien, ahora hay que ver si se llega al borde de la ventana. Hay que
    # recordar que el centro tiene coordenadas x = 0 e y = 0. Así que
    # si se pasa del borde derecho de la ventana (x = 320) o del borde
    # izquierdo (x = -320) hay que cambiar el sentido de movimiento
    if self.x > 320 or self.x < -320:
        # Al cambiar el signo de la velocidad, se consigue lo que queremos
        self.dx = -self.dx
    # Y hacemos lo propio con los bordes superior e inferior...
    if self.y > 240 or self.y < -240:
        self.dy = -self.dy

def crear_punto(evento):
    Punto(evento.x, evento.y)

pilas.eventos.click_de_mouse.conectar(crear_punto)

pilas.ejecutar()
```

## Programa 6: cuadradosEnGrupo.py

Finalmente, vamos a ver cómo podemos actuar con todos los puntos a la vez aprovechando el concepto de **Grupos** de Pilas...

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

import pilas
from random import randint

pilas.iniciar()

pilas.fondos.Color(pilas.colores.negro)

class Punto(pilas.actores.Actor):

    def __init__(self, x=0, y=0):
        imagen = pilas.imagenes.cargar_superficie(4, 4)
```

```
color = pilas.colores.Color(randint(0, 255), randint(0, 255),
                             randint(0, 255))
imagen.pintar(color)
pilas.actores.Actor.__init__(self, imagen)
self.x = x
self.y = y
self.dx = 1
self.dy = 1

def actualizar(self):
    self.x += self.dx
    if self.x > 320 or self.x < -320:
        self.dx = -self.dx
    self.y += self.dy
    if self.y > 240 or self.y < -240:
        self.dy = -self.dy

# Hasta ahora, hemos creado los puntos sin más. Si luego queremos hacer algo
# con ellos, sería más interesante guardar un listado de lo creado. Para ello
# es, precisamente, el concepto de Grupo. Vamos a crear, inicialmente, un grupo
# vacío que luego contendrá a nuestros puntos.
puntos = pilas.grupo.Grupo()

def crear_punto(evento):
    # Ahora, además de crear un punto al hacer click con el ratón como hacíamos
    # antes, lo añadimos al grupo
    puntos.append(Punto(evento.x, evento.y))

# Vamos a mostrar una utilidad sencilla de tener la referencia de todos los
# puntos creados. Vamos a definir una función que reaccione al pulsar una tecla
# con todos los puntos a la vez.
def reaccionar_tecla(evento):
    # Si se ha pulsado la tecla 'x', ponemos la coordenada x de todos los
    # puntos a cero. Fíjate lo fácil que es hacerlo con la referencia del grupo
    if evento.codigo == 'x':
        puntos.x = 0
    # Si la tecla es 'y', hacemos lo propio con la coordenada y
    elif evento.codigo == 'y':
        puntos.y = 0

pilas.eventos.click_de_mouse.conectar(crear_punto)

# Finalmente, falta conectar la nueva función con el evento de pulsar la tecla
pilas.eventos.pulsa_tecla.conectar(reaccionar_tecla)

pilas.ejecutar()
```

¿Qué es lo que tienes que hacer? Escribe cada uno de los programas, comprueba que funcionan y **envíalos todos juntos** a tu profesor. Para subir nota, como programa en un correo aparte, podrías hacer dos modificaciones más. En primer lugar, ¿sabrías conseguir que los puntos se movieran más deprisa? Hay dos formas, una de ellas muy simple, la otra requiere de algo de investigación. Y en segundo lugar, más complicado, ¿sabrías cambiar el color de cada punto en cada fotograma?...

---

## Recapitulación

Esta práctica has visto cómo funciona la animación a base de fotogramas, has manejado las posiciones de los protagonistas mediante coordenadas y has trabajado con **clases**, **objetos** y **grupos**. Muchos conceptos, sí. Recuerda que puedes consultar la documentación y la wiki para encontrar más explicaciones.

- ¿Controlas los diferentes tipos de **eventos** que hemos manejado?
- ¿Hay visto las dos formas diferentes de emplear **import**? ¿Cómo afecta a **randint()**?
- ¿Cómo se hace para crear una clase? ¿Y para crear una clase que derive de otra?
- ¿Para qué se usa **\_\_init\_\_()**? ¿Hay otros métodos especiales?
- ¿Para qué se usa **self**? ¿Qué diferencia **self.x** y **x**, por ejemplo?
- ¿Qué representa el atributo **codigo** del evento? ¿Qué otros eventos y atributos hay?