Tutorial Pong-C&A Documentation

Release 0.1

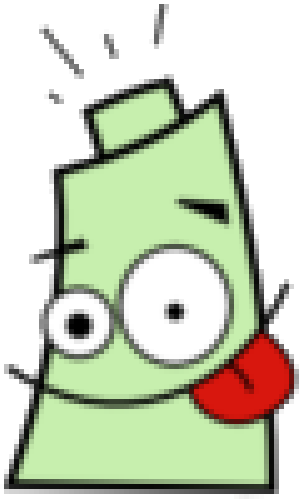
Luciano Castillo

April 05, 2016

Contents

1 Índice: **3**
- 1.1 Introducción . . . 3
- 1.2 Actores y fondos . . . 3
- 1.3 Menú de opciones . . . 7
- 1.4 Escenas: menú y controles . . . 9
- 1.5 Escena juego . . . 13

“Programando con pilas-engine”



Pilas

Simplificando el desarrollo de video juegos.

CHAPTER 1

Índice:

1.1 Introducción

Este tutorial se creó para enseñar algunos pasos básicos en la creación de un juego 2d, usando pilas-engine (para ver como se instala, hacé click acá). Además, en el transcurso del mismo, vamos a hacer nuestro propio juego, el "Pong".

Para empezar a usar pilas-engine, podemos ver el manual de uso (o descargarlo), para tener algunas nociones básicas. También se puede consultar a este resumen de pilas. En este caso empezaremos viendo qué son los actores y los fondos.

1.2 Actores y fondos

1.2.1 Actores

Un concepto importante en pilas es del de actores.

Actor:

Un actor en pilas **es un objeto que aparece en pantalla, tiene una posición determinada y se puede manipular**.

Por ejemplo:

- Una nave.
- Un ingeniero.
- Una zanahoria.



Para empezar, pilas se puede usar directamente desde un intérprete interactivo de python. Una vez dentro del intérprete, tienes que escribir estas dos líneas de código:

```
1 import pilas          # importa la librería pilas
2 pilas.iniciar()       # inicia una ventana predeterminada
```

En tu pantalla tiene que aparecer una ventana de color gris.

Para ver un ejemplo de actor podemos ingresar (luego de haber generado la vetana con las líneas anteriores):

```
bomba = pilas.actores.Bomba()          # le asignamos a "bomba" el actor "Bomba()", ubicado en pilas.a
```

Se va a crear inmediatamente una bomba, en la ventana, como ésta:

Como Bomba es un actor, encontraremos mucha funcionalidad en él que la tendrán el resto de los actores.

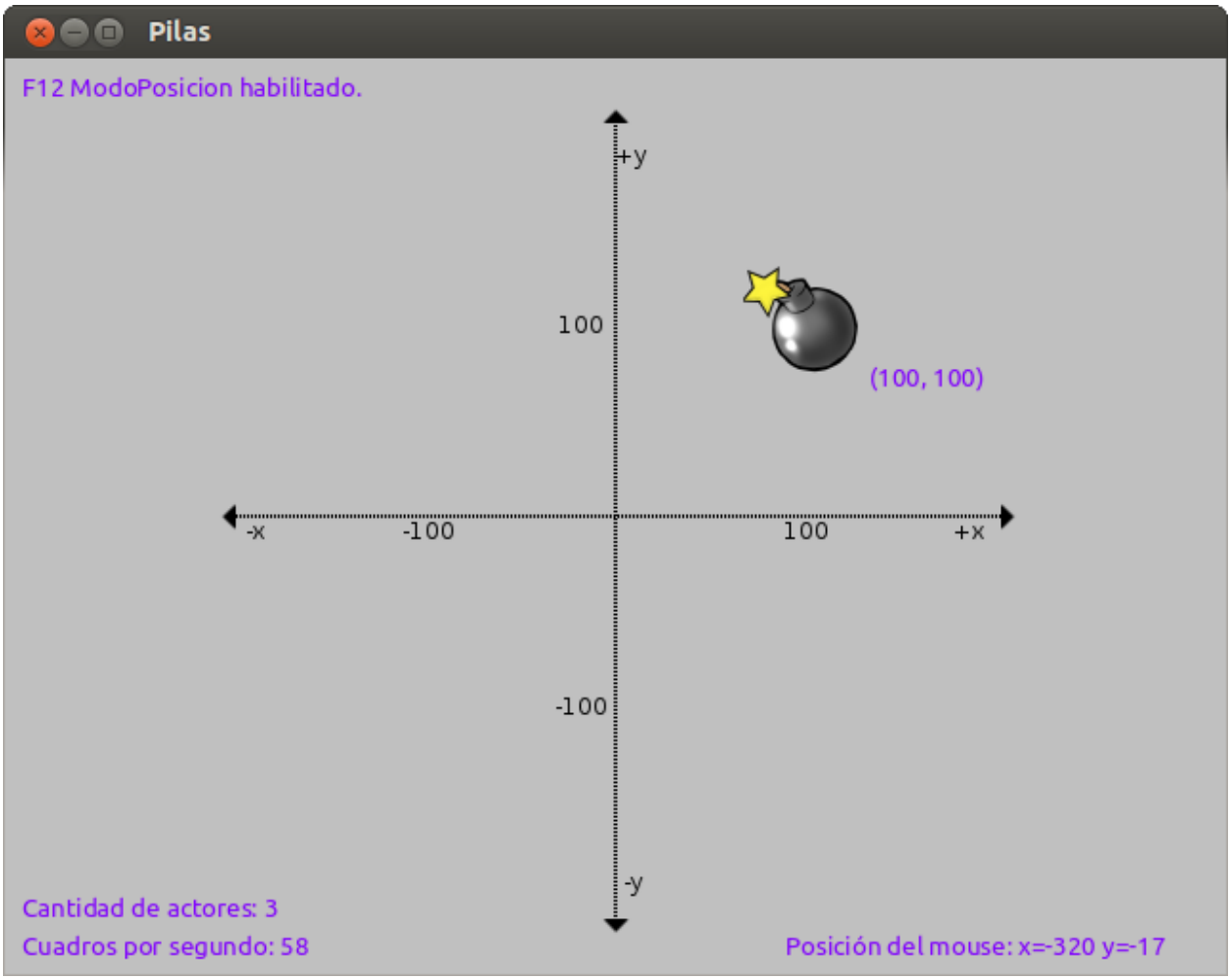
Por ejemplo:

Posición

Podemos cambiar la posición del actor en el eje de cordenadas mediante las propiedades x e y:

```
bomba.x = 100
bomba.y = 100
```

Se puede observar si pulsas la tecla F12, la posición de los actores en el eje, de esta forma:



Escala

Este atributo indica su tamaño en pantalla:

```
1 bomba.escala = 2          # le duplicamos el tamaño a "bomba"
```

Rotación

La rotación siempre se indica en grados, e indica el grado de inclinación hacia la derecha:

```
bomba.rotacion = 45
```

Eliminar actor

Para eliminar un actor basta con llamar a la función eliminar():

```
bomba.eliminar()
```

1.2.2 Fondos

Otro concepto a aprender es el de fondos.

Fondo:

En pilas **un fondo es** un concepto muy simple, es solamente **una imágen detras de mis actores, que suele ser un paisaje**.

Por ejemplo:

```
fondo = pilas.fondos.Tarde()
```

Bueno, después de esta primera parte ya vamos a poder probar muchas cosas, pero ya que esta guía está destinada a un determinado juego, para conocer más y poder aplicarlo consulten al manual.

A partir de ahora solo vamos a ver lo necesario para nuestro "Pong".

1.3 Menú de opciones

Note: **En un juego es fundamental un menú de opciones** (para jugar, salir, consultar los controles, etc.), es por eso que pilas tiene destinado un actor "Menu()" (el que vamos a usar).

Warning: Es importante que sepan no hace falta que el juego salga exactamente igual al del tutorial, es más, si le quieren hacer modificaciones porque algo no les gusta háganlo, eso les va a ayudar a conocer más sobre pilas y a personalizar el juego.

Bueno, para generar un menú hay que pasarle una lista de opciones; en nuestro caso serán "Empezar juego", "Ver controles" y "Salir". Pero también se deben definir funciones que nos dirán que pasará al seleccionar esa opción. Por ejemplo:

```
import pilas          # importa la librería pilas

pilas.iniciar()       # inicia la ventana

def selecciona_empezar():   # función que se ejecutará al seleccionar Empezar
    pilas.avisar('Ha seleccionado "Empezar"') # muestra en pantalla que se ha seleccionado Empezar

def selecciona_salir():   # función que se ejecutará al seleccionar Salir
    pilas.avisar('Ha seleccionado "Salir"') # muestra en pantalla que se ha seleccionado Salir

opciones = [('Empezar', selecciona_empezar),  # relaciona el texto "Empezar" con su función seleccion
            ('Salir', selecciona_salir)]      # relaciona el texto "Salir" con su función selecciona_

menu = pilas.actores.Menu(opciones)    # creás un actor "menu" de tipo "Menu()"; y le pasás las opci

pilas.ejecutar()      # ejecuta las líneas escritas anteriormente
```

Como hay muchas cosas nuevas, vamos a ver paso a paso lo que hicimos:

1. Primero importamos pilas.
2. Luego abrimos una ventana.
3. Definimos la función "**selecciona_empezar**":
   - "`pilas.avisar('Ha seleccionado "Empezar"')`" genera un texto que aparecerá abajo y a la izquierda de nuestra ventana. En este caso: 'Ha seleccionado "Empezar"'.
4. Definimos la función "**selecciona_salir**":
   - "`pilas.avisar('Ha seleccionado "Salir"')`" genera un texto que aparecerá abajo y a la izquierda de nuestra ventana. En este caso: 'Ha seleccionado "Salir"'.
5. Generamos una variable "opciones" en donde vamos a listar todas las opciones de nuestro menú junto con el nombre de la función que se ejecutará al hacerle click.
6. Creamos una variable "menu" donde le asignamos el actor "pilas.actores.Menu(opciones)". Éste actor tiene la particularidad de recibir como único parámetro una lista (la del punto anterior, "opciones"), donde estarán todas las opciones posibles.
7. Y por último ponemos "pilas.ejecutar()". Ésta última línea solo se pone cuando escribimos el código en un archivo para ejecutarlo, en la consola interactiva no hace falta.

Nos debería quedar algo así:

Para que el menú se vea mejor, podríamos ponerle un fondo y un título:

Para poner el fondo (que se lo pueden bajar de acá o cambiarlo), recordamos:

```
fondo = pilas.fondos.Fondo('pilas-fondo-final.png')      # esta función toma la ruta de una imagen y
```

Y para el título (lo pueden bajar de acá o cambiarlo), se puede generar un actor con esa imágen, así:

```
titulo = pilas.actores.Actor('pilas-titulo-final.png')      # esta función toma la ruta de una image
```

1.4 Escenas: menú y controles

1.4.1 "Salir"

Ahora que ya tenemos un menú bien hecho podemos seguir con las funciones que implica. Empecemos con la más fácil, "Salir":

```
def selecciona_salir():
    pilas.terminar()        # utilizamos esta funcion de pilas para cerrar nuestra ventana
```

1.4.2 Escenas

La siguiente función va a estar destinada a los controles. Para poder hacerla hay que empezar a trabajar con escenas.

Escena:

Las escenas en pilas **son las partes de un juego**, cuando termina una escena se borran todos sus actores y comienza una nueva con actores nuevos.

Cuando ponemos pilas.iniciar() se genera una escena llamada Normal, esta escena no tiene un comportamiento muy elaborado, simplemente imprime toda la pantalla de gris para que podamos colocar actores sobre ella y veamos una escena limpia.

Para generar una escena simplemente hay que meter todos sus actores y fondos en una clase, al ejecutar cada una de ellas automáticamente se borrarán los actores y fondos de escenas anteriores.

Veamos un ejemplo:

```
class PantallaBienvenida(pilas.escenas.Escena):      # definimos una clase para la escena
    def __init__(self):
        pilas.escenas.Escena.__init__(self)
        fondo = pilas.fondos.Pasto()      # generamos un fondo
        texto = pilas.actores.Texto('Bienvenido a pilas!!!')      # generamos un actor texto

PantallaBienvenida()      # ejecutamos la clase (escena)
```

Se nos va a crear algo como esto:

Pero si generamos otra clase, se va a reiniciar la escena:

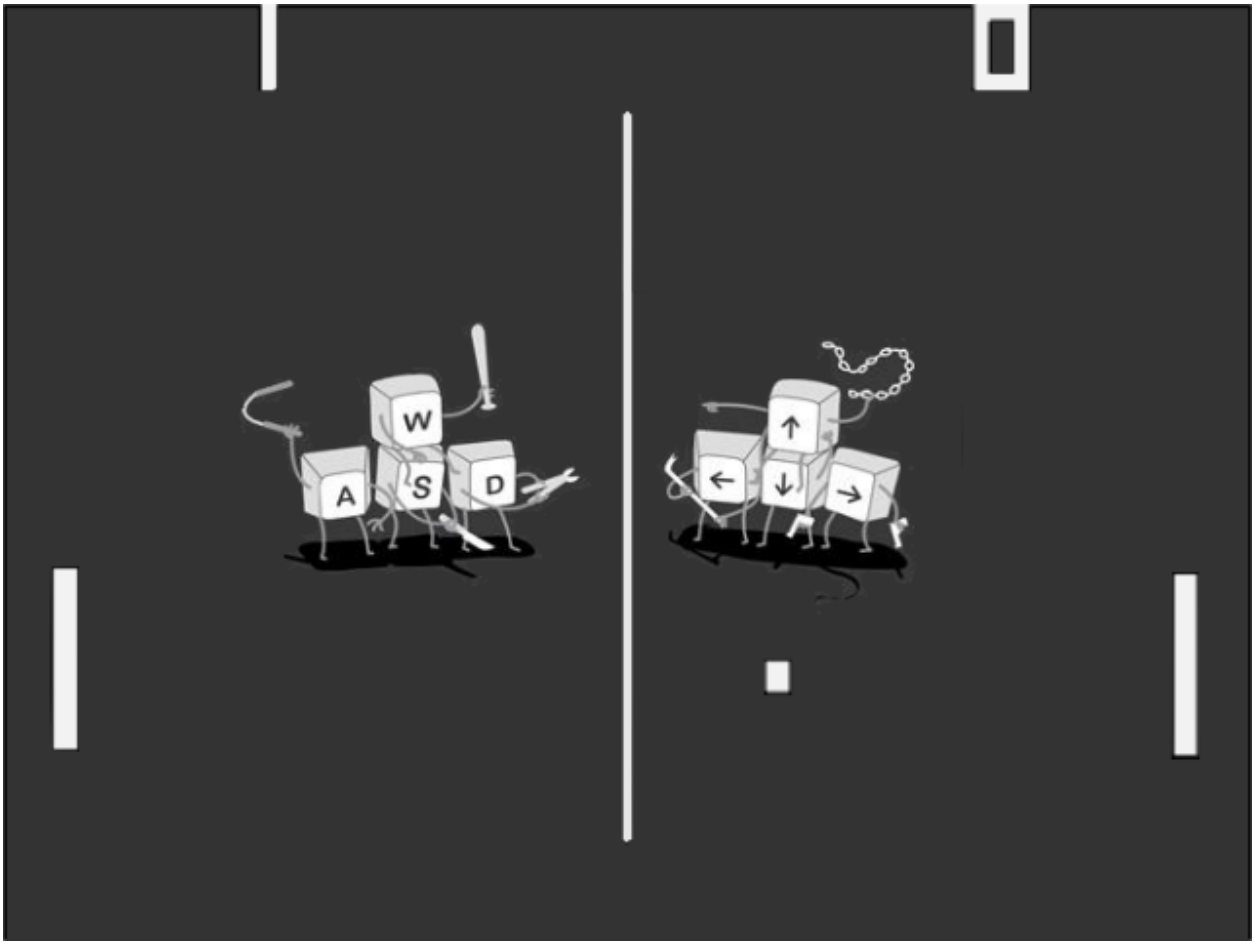
```
class PantallaBienvenida2(pilas.escenas.Escena):      # definimos una clase para la escena
    def __init__(self):
        pilas.escenas.Escena.__init__(self)
        fondo = pilas.fondos.Tarde()      # generamos un fondo
        texto = pilas.actores.Texto('Entraste a pilas!!!')      # generamos un actor texto

PantallaBienvenida2()      # ejecutamos la clase (escena)
```

Y va a pasar algo como esto:

Así como cada actor tiene su lugar en "pilas.actores.*" , para definir una escena hace falta pasarle de parámetro a la clase, "`pilas.escenas.Escena`", eso hará que la clase sea una escena y no un actor. Más adelante veremos como hacer clases para cada actor.

1.4.3 Escena controles

Para continuar con la escena controles, yo usé de fondo la siguiente imagen:



Note: Acuérdense que pueden cambiar las imágenes usadas, como también algunos detalles, para hacer su propia versión.

Siguiendo con el código de nuestro programa, habrá que implementar una escena nueva, por lo tanto habrá que crear una clase:

```
class Escena_controles(pilas.escenas.Escena):      # definimos la clase Escena_controles (Escena)
    def __init__(self):
        pilas.escenas.Escena.__init__(self)
        fondo = pilas.fondos.Fondo('data/fondo_ayuda.png')      # le damos un fondo
        titulo = pilas.actores.Texto('Controles', magnitud=30, y=200)      # colocamos el titulo
        titulo.color = pilas.colores.rojo      # le damos un color
        texto1 = pilas.actores.Texto('Jugador 1 (rojo)', y=120, x=-120)      # colocamos subtitulo
        texto2 = pilas.actores.Texto('Jugador 1 (rojo)', y=120, x=120)      # colocamos subtitulo
        pilas.avisar('Pulsa ESC para regresar al menu')
```

Lo que hicimos fue simplemente declarar una clase para crear una escena nueva, le dimos un fondo y dos títulos simbolizando a cada jugador.

Si todo salió bien, debería quedar así:

Bueno, habiendo terminado estas dos funciones, solo nos queda ver la que está vinculada a la escena del juego.

Para continuar, debemos tener en cuenta que esta escena es la mas extensa y complicada, ya que es donde interactúan todos los actores del juego.

1.5 Escena juego

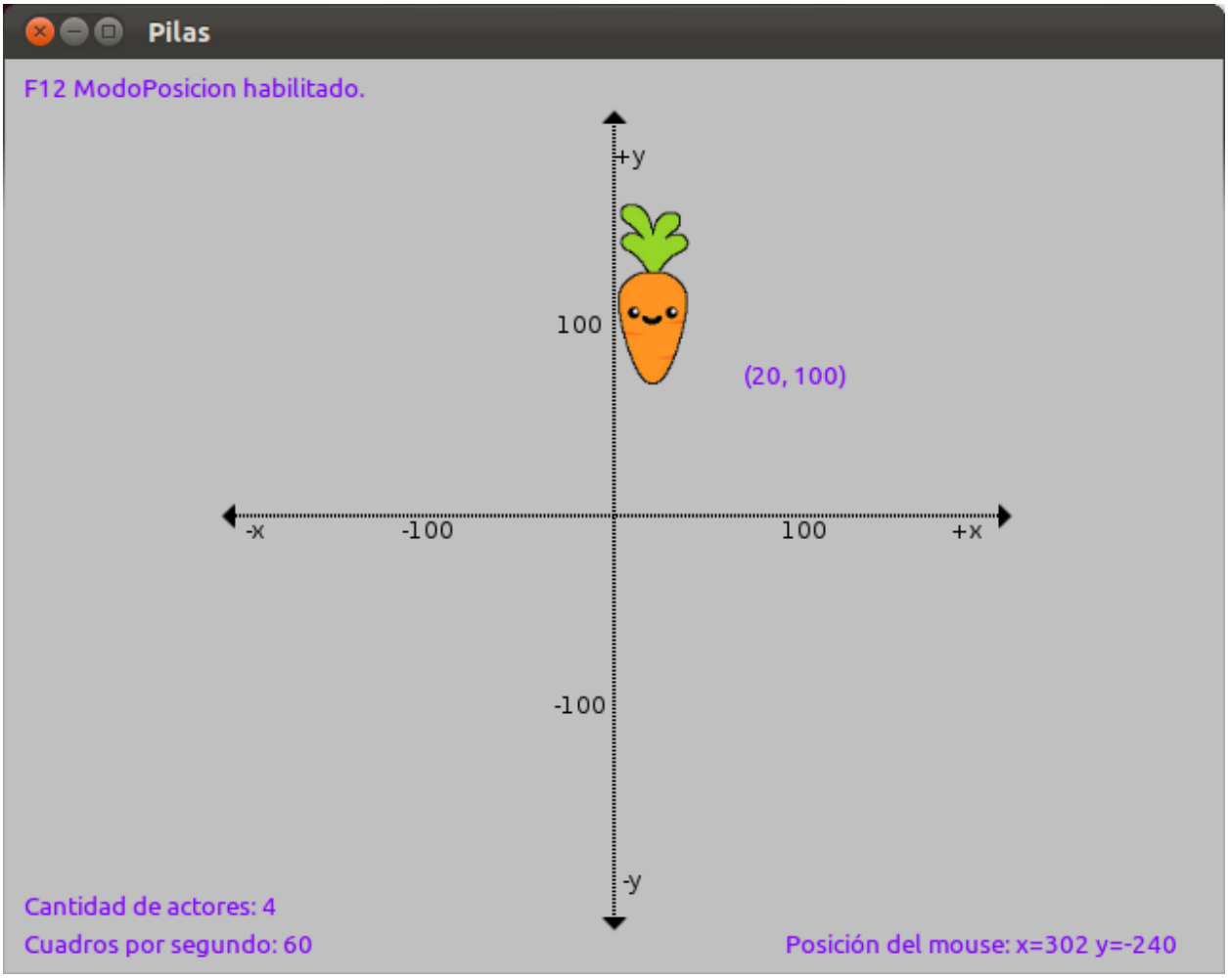
1.5.1 Clases de actores

Vamos a empezar creando los actores con sus distintas propiedades y habilidades, para eso vamos a crear un archivo aparte llamado "actores.py". Dentro de él haremos una clase para cada actor.

Un ejemplo de clase para un actor sería:

```
class Zanahoria(pilas.actores.Zanahoria):
    def __init__(self, x=0, y=0):
        pilas.actores.Zanahoria.__init__(self, x=x, y=y)
        self.aprender(pilas.habilidades.Arrastrable)

Zanahoria(20, 100)
```

Esto quedaría así:



Conociendo como es el sistema para definir actores en clases, continuamos con el código del "Pong".

Para la pelota del juego decidimos usar una bomba ("pilas.actores.Bomba()"), que explota al ganar uno de los dos jugadores:

```
class Pelota(pilas.actores.Bomba):      # representa la pelota del juego
    def __init__(self, x=0, y=0):
        pilas.actores.Bomba.__init__(self, x=x, y=y)
        self.circulo(x, y) = pilas.fisica.Circulo(x, y, 20, restitucion=1, friccion=0, amortiguacion=
        self.imitar(self.circulo)      # hacemos que la pelota siga los movimientos del circulo de fi
        self.dx = 1      # creamos coeficiente de impulso para x
        self.dy = 1      # creamos coeficiente de impulso para y
        self.circulo.impulsar(self.dx * 50000, self.dy * 50000)      # impulsamos por primera vez la
```

Los actores que usaremos en nuestro programa serán:

1. Jugador 1
2. Jugador 2
3. Puntajes
4. Pelota

Teniendo ya creada la pelota del juego, continuaremos con los jugadores. Las imágenes usadas para la creación de los mismos son:

Comenzando con el jugador 1, haremos:

```
class Jugador1(pilas.actores.Actor):      # definimos la clase del jugador 1
    def __init__(self, x=0, y=0):
        pilas.actores.Actor.__init__(self, x=x, y=y)
        self.imagen = pilas.imagenes.cargar('BaseRoja.bmp')      # le cargamos la imagen
        self.radio_de_colision = 15      # le asignamos el radio de colision al actor
        self.aprender(pilas.habilidades.SeMantieneEnPantalla)      # le enseñamos a mantenerse en pan
        self.aprender(self.MoverseConWS)      # le enseñamos a moverse con W y S, con una habilidad c
```

Basicamente, lo que hicimos fue cargar una imagen (llamada “BaseRoja.bmp” en este caso), asignarle un radio de colisión, obligarlo a mantenerse en pantalla y brindarle movimiento con las teclas W y S.

Para el jugador 2 es basicamente lo mismo, salvo que en vez de aprender a moverse con W y S, se mueve con las teclas Arriba y Abajo.

A continuación les mostramos el código:

```
class Jugador2(pilas.actores.Actor):      # definimos la clase del jugador 2
    def __init__(self, x=0, y=0):
        pilas.actores.Actor.__init__(self, x=x, y=y)
        self.imagen = pilas.imagenes.cargar('BaseAzul.bmp')      # le cargamos la imagen
        self.radio_de_colision = 15      # le asignamos el radio de colision al actor
        self.aprender(pilas.habilidades.SeMantieneEnPantalla)      # le enseñamos a mantenerse en pan
        self.aprender(self.MoverseConArribaAbajo)      # le enseñamos a moverse con las teclas arriba

    class MoverseConArribaAbajo(pilas.habilidades.Habilidad):      # esta clase define la habilidad
        def __init__(self, receptor):
            pilas.habilidades.Habilidad.__init__(self, receptor)
            pilas.eventos.actualizar.conectar(self.pulsa_tecla)      # le decimos que actualice la fu

        def pulsa_tecla(self, evento):
            velocidad = 5
            c = pilas.mundo.control

            if c.arriba:    # si se ha pulsado arriba
                self.receptor.y += velocidad     # movemos al receptor (actor) para arriba
```

```
            if c.abajo:     # si se ha pulsado abajo
                self.receptor.y -= velocidad     # movemos al receptor (actor) para abajo
```

Por último, necesitamos puntajes, para esto deberán definir dos clases muy simples:

```
class Puntaje1(pilas.actores.Puntaje):      # definimos la clase del puntaje 1
    def __init__(self, x=-70, y=220):
        pilas.actores.Puntaje.__init__(self, x=x, y=y)
        self.color = pilas.colores.blanco      # le cargamos el color

class Puntaje2(pilas.actores.Puntaje):      # definimos la clase del puntaje 2
    def __init__(self, x=70, y=220):
        pilas.actores.Puntaje.__init__(self, x=x, y=y)
        self.color = pilas.colores.blanco      # le cargamos el color
```

1.5.2 Habilidades

Habilidad:

Una habilidad tambien **es una clase**, que está aplicada a un actor, y que no se ve por si sola.
La habilidad le enseña comportamientos al actor para diferenciarse de otros de su misma especie.

Por ejemplo, si creamos un mono llamado mono y le enseñamos la habilidad "pilas.habilidades.Arrastrable":

```
mono = pilas.actores.Mono()      # creamos al mono
mono.aprender(pilas.habilidades.Arrastrable)     # le enseñamos que sea arrastrable
```

Este mono será distinto a los demás ya que se puede arrastrar con el mouse, en este caso la habilidad que usamos ya estaba creada en pilas, pero si queremos que un actor haga cosas diferentes y no encontramos una habilidad adecuada, podemos crear nuestras propias habilidades.

Para crear habilidades, lo único que hay que tener en cuenta es que la clase no va a ser de tipo "pilas.actores.Actor", ni "pilas.escenas.Escena", sino que va a ser una habilidad de tipo "pilas.habilidades.Habilidad".

Comencemos con el jugador 1 o jugador rojo, para crearlo necesitamos crear una nueva habilidad que le va a enseñar a nuestro actor a moverse con las teclas <W> y <S>, para ir arriba y abajo.

A continuación veremos un ejemplo sencillo de habilidad creada por nosotros y aplicable a nuestro programa, la cual **debe estar definida dentro de la clase Jugador1**:

```
from pilas.simbolos import *

class MoverseConWS(pilas.habilidades.Habilidad):      # definimos la habilidad para que se mueva con
    def __init__(self, receptor):      # ademas de su clase necesita saber cual es su receptor (el a
        pilas.habilidades.Habilidad.__init__(self, receptor)
        self.w = False      # esta variable sera True cuando se pulse la tecla w
        self.s = False      # esta variable sera True cuando se pulse la tecla s
        pilas.eventos.actualizar.conectar(self.pulsa_tecla)      # le decimos que actualice 'pulsa_te
        pilas.eventos.pulsa_tecla.conectar(self.cuando_pulsa_la_tecla)
        pilas.eventos.suelta_tecla.conectar(self.cuando_suelta_la_tecla)

    def pulsa_tecla(self, evento):      # esta funcion mueve el actor para arriba (si pulso w) o aba
        velocidad = 5
        if self.w:
            self.receptor.y += velocidad
        elif self.s:
```

```
            self.receptor.y -= velocidad

    def cuando_pulsa_la_tecla(self, evento):      # detecta si pulso una tecla y cambia el estado de
        self.procesar_cambio_de_estado_en_la_tecla(evento.codigo, True)

    def cuando_suelta_la_tecla(self, evento):      # detecta si solto una tecla y cambia el estado de
        self.procesar_cambio_de_estado_en_la_tecla(evento.codigo, False)

    def procesar_cambio_de_estado_en_la_tecla(self, codigo, estado):     # recibe y cambia el estado
        mapa = {w: 'w', s: 's'}
        if mapa.has_key(codigo):
            setattr(self, mapa[codigo], estado)
```

Bueno, en esta clase podemos encontrar muchas cosas nuevas y que van a parecer difíciles de entender, pero las vamos a ver de nuevo, una por una, para saber como funcionan.

Empecemos con la función "pilas.eventos.actualizar.conectar()". Lo único que hace esta función es recibir otra, definida dentro de la clase (por eso lleva "self." adelante), y ejecutarla constantemente (actualizarla).

Por lo general, se usa para comprobar una condición que cambia su estado (puede ser "True" o "False", en este caso) durante el desarrollo del juego.

Lo que hicimos dentro de esta función que pasamos como parámetro ("self.pulsa_tecla"), fue:

1. Definir una velocidad.
2. Comprobar si se ha pulsado w, si así fue, sumar 5 (la velocidad) a la posición dentro de la coordenada "y" (hacia arriba) del actor (receptor).
3. De lo contrario, si se pulso s, en este caso, restar la velocidad a la posición "y" del actor (para que este vaya hacia abajo)

Para saber si se ha pulsado o soltado una de esas teclas usamos las variables "self.w" y "self.s", pero necesitamos que cambien su estado (valor: "True" o "False"), dependiendo de la interacción del usuario. Para esto, utilizamos la siguiente funciones: "pilas.eventos.pulsa_tecla.conectar()" y"pilas.eventos.suelta_tecla.conectar()"; que como nos dicen sus nombres, detectan la pulsación y despulsación de alguna tecla.

> **Warning:** Hay que saber distinguir entre la función: "self.pulsa_tecla()", **definida por nosotros** en la clase (por eso el "self."); y la otra función totalmente distinta: "pilas.eventos.pulsa_tecla.conectar()", definida en el módulo pilas y **llamada por nosotros**.

Volviendo con las funciones para detectar la pulsación de teclas, las dos tienen algo en común: llaman a otra función (creada por nosotros), "self.procesar_cambio_de_estado_en_la_tecla()", que recibe un codigo (la tecla que se pulsó) y un estado ("True" si se pulsó, "False" si se soltó). Esta función lo que hace es:

1. Crear un mapa con las posibilidades de teclas que se puedan haber pulsado (en este caso solo nesecitamos W y S).
2. Preguntar si el codigo recibido pertenece a alguna opción del mapa ("mapa.has_key(codigo)", ¿mapa contiene una clave <W o S según sea>?).
3. Y por último usa una función de python llamada "setattr()", que recibe tres parámetros y funciona más o menos así:

```
>>> setattr(parametro1, parametro2, parametro3)
parametro1.parametro2 = parametro3
```

Lo que hace es simplemente setear atributos, como lo dice el nombre. Se le pasa un 1° parámetro que va a ser la clase, el 2° que va a ser el atributo y el 3° que va ser el valor que se le asigne.

Para terminar de entenderlo mejor veamos nuestro caso, haciendo de cuenta que la tecla pulsada (si es pulsada la tecla, el estado va a ser "True") fue W:

```
>>> setattr(self, mapa[codigo], estado)
self.w = True
```

Ahora se entiende por que declaramos las variables "self.w" y "self.s". Para conocer el estado de cada una en la función "self.pulsa_tecla()".

Habiendo creado ya las distintas escenas con sus respectivos fondos y actores. Nuestro juego está casi terminado, solo nos quedaría crear la parte de las colisiones (para que cuando la pelota toque un lado, se sume un punto en el puntaje; y para que a determinada cantidad de puntos explote la bomba y termine el juego).

En este momento nuestro "Pong" se vería así:

1.5.3 Colisiones

La parte de colisiones aun no está desarrollada, este tutorial será actualizado muy pronto.

Note: Recuerden que los puntajes deben cambiar su valor a medida que tocan los laterales; así como también, que deben crear una colisión para que la bomba rebote en las dos barras.

Acá se encuentra el código del "Pong", junto con las imágenes usadas y este tutorial.

Cualquier contribución será bien aceptada, ¡Suerte!