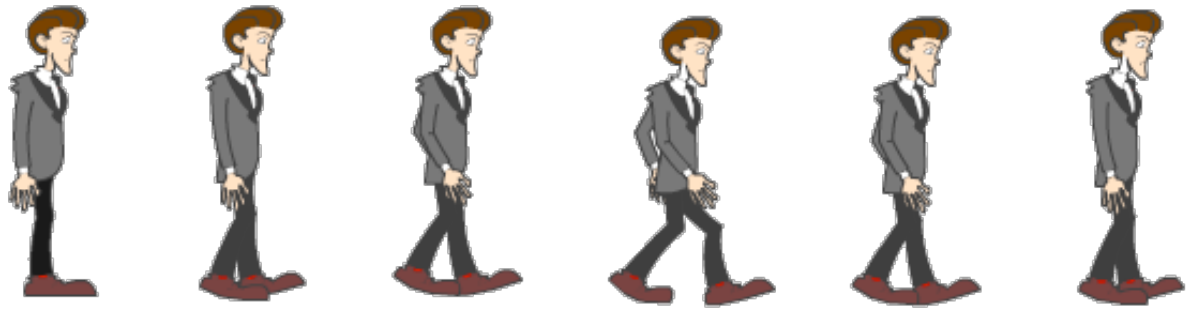
Grillas y Animación

En el manual y en los ejemplos de **Pilas**, puedes ver cómo usar **grillas** para tener tus personajes animados de forma automática. Pero el tema es algo más complicado cuando quieres hacer que tengan comportamientos específicos que se coordinen con los diferentes fotogramas que forman parte del diseño. Vamos a ver cómo hacer esto, paso a paso.



Partiremos de un personaje (llamémoslo **chuck**) cuyo diseño descansa en la grilla superior; tenemos un primer fotograma o **cuadro** (en el lenguaje de Pilas) en posición de descanso, y otros 5 cuadros que muestran el movimiento del hombre andando. Éste es el planning que vamos a seguir:

- **andando01.py**
  Usaremos el primer cuadro y lo desplazaremos por la pantalla con los controles por defecto que incorpora pilas (las flechas del cursor).
- **andando02.py**
  Aprenderemos a desplazar al actor con las teclas que queramos, usando el módulo **control**.
- **andando03.py**
  Haremos que en el movimiento, el actor cambie de cuadro simulando el caminar. Lo haremos manualmente y veremos sus limitaciones.
- **andando04.py**
  Aprenderemos de la experiencia de Pilas y de los actores que incorpora, usando el módulo **comportamientos** y consiguiendo una animación más real y extensible.
- **andando05.py**
  Extenderemos el comportamiento del actor para que también salte, viendo las ventajas que derivan del planteamiento del paso anterior.

¡Vamos allá!

andando01.py

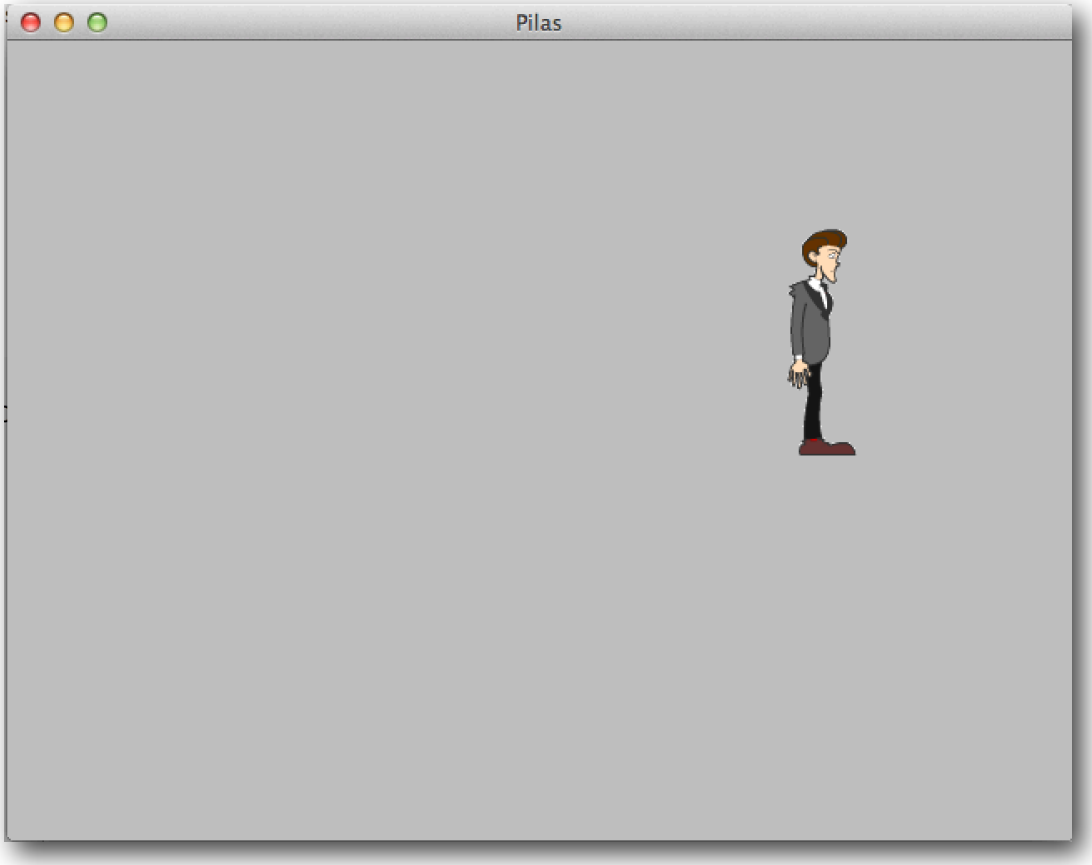
```
#! /usr/bin/env python
# -*- coding: utf-8 -*-
#---------------------------------------------------------------------
# andando01.py
# Movimiento Automático
#---------------------------------------------------------------------

import pilas

pilas.iniciar()


# Definimos la clase de nuestro actor
class Hombre(pilas.actores.Actor):
    "Un actor que se mueve con el teclado"

    def __init__(self):
        pilas.actores.Actor.__init__(self)
        self.imagen = pilas.imagenes.cargar_grilla("andando.png", 6)
        self.aprender(pilas.habilidades.MoverseConElTeclado)

chuck = Hombre()

pilas.ejecutar()
```

Llegados a este punto, este paso tendría que estar claro para ti (si no lo has hecho todavía, tendrías que leer el tutorial de Mario). Hemos definido la clase de actor **Hombre** y luego la hemos **instanciado**, creando un actor de ese tipo al que hemos llamado, **chuck**.

Dos matices nuevos. Primero, la imagen de chuck la cargamos usando el método **cargar_grilla()**. Con éste método hemos de indicar el número de cuadros que forman la grilla (en nuestro caso, **6**). Es muy importante el que tengamos en cuenta que Pilas divide entonces la imagen de referencia en un mosaico de piezas iguales para usarlas; nuestros cuadros han de ser todos del mismo tamaño.

Por otra parte, conseguir que un actor se mueva con los controles que proporciona Pilas por defecto es muy sencillo. Basta usar el método **aprender()** del actor y pasarle como argumento la habilidad **pilas.habilidades.MoverseConElTeclado**. A partir de ese momento, nuestro chuck se moverá con las flechas del cursor (izquierda, derecha, arriba y abajo)

Prueba (da al código el nombre **andando01.py**) y ejecútalo. Y si quieres aprender más del movimiento por defecto con el teclado, usa **dir()**, **help()** o **pilas.ver()** en el intérprete. ¿Lo recuerdas?



Hay otras habilidades que podrías usar con el actor. Por ejemplo, cuando lo mueves, es fácil que se salga de la pantalla. Puedes controlarlo a partir de las coordenadas y del tamaño de la ventana, pero hay una forma automática para ello, y es aprender la habilidad **pilas.habilidades.SeMantieneEnPantalla**; una vez hecho esto, por mucho que lo intentes, chuck se mantendrá siempre dentro de los límites de la ventana. De nuevo, siéntete libre de husmear por las tripas del módulo **pilas.habilidades**.

Bueno. Y ¿qué sucede si queremos manejar a nuestro personaje con otras teclas diferentes? Avancemos al siguiente paso, en el que usaremos las teclas **a** y **s** para mover al actor hacia la izquierda y la derecha, respectivamente.

andando02.py

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-
#---------------------------------------------------------------------
# andando02.py
# Movimiento con controles propios
#---------------------------------------------------------------------

import pilas

pilas.iniciar()

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

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


# Definimos la clase de nuestro actor
class Hombre(pilas.actores.Actor):
    "Un actor que se mueve con las teclas a y s"

    def __init__(self):
        pilas.actores.Actor.__init__(self)
        self.imagen = pilas.imagenes.cargar_grilla("andando.png", 6)
        # Hacemos que el actor se mueva con nuestro control personalizado
        self.aprender(pilas.habilidades.MoverseConElTeclado,
                      control=mandos)

chuck = Hombre()

pilas.ejecutar()
```

(Como verás, indicamos los cambios que hemos realizado en negrita y con fondo gris, al tiempo que lo que no ha cambiado lo hemos dejado sin colorear)

Lo primero que notarás es la siguiente línea:

```
teclas = {pilas.simbolos.a: 'izquierda', pilas.simbolos.s: 'derecha'}
```

La variable **teclas** es un **diccionario**. Ésta es la manera que usa Pilas para definir controles. Los diccionarios están formados por parejas; la llamada **clave** y su **valor** correspondiente. En el caso de los controles de Pilas, **en la clave indicaremos la tecla**

que deseemos mientras que **en el valor pondremos la dirección** del movimiento. No nos cansaremos de insistir: usa **help(pilas.simbolos)**, por ejemplo, para ver las teclas predefinidas. En nuestro caso, vamos a usar la tecla **a** para el movimiento hacia la izquierda y la tecla **s** para la derecha.
Lo siguiente es crean un objeto que controle esas teclas y que nos permita utilizarlo para manejar a chuck. Eso lo logramos con la siguiente línea:

```
mandos = pilas.control.Control(pilas.escena_actual(), teclas)
```

Los objetos de la clase **pilas.control.Control** se encargan, precisamente, de la tarea que deseamos. Al constructor de la clase, como vemos, **sólo tenemos que pasarle dos argumentos**: El primero es **la escena** para la que vamos a usarlo (en nuestro ejemplo, tenemos una sola escena y de ahí que usemos **pilas.escena_actual()**) y el segundo es el diccionario que hemos definido con **las teclas** que vamos a usar.

Una vez que tenemos el **nuevo control** definido, al que hemos llamado **mandos**, podemos usarlo con el actor que queramos. ¿Recuerdas que en el paso anterior, para usar las teclas que trae por defecto Pilas, hemos usando en la definición de chuck lo siguiente?

```
self.aprender(pilas.habilidades.MoverseConElTeclado)
```

Bien, pues esa línea la hemos de cambiar por esta otra:

```
self.aprender(pilas.habilidades.MoverseConElTeclado,
                    control=mandos)
```

¿Ves qué simple? Simplemente hemos de incluir un argumento más, llamado **control**, con el objeto que contiene el control de teclado que hemos definido (nuestro **mandos**).

¡Pruébalo! Si ejecutas el código anterior con el nombre **andando02.py**, verás que ahora puedes mover a chuck con las teclas **a** y **s** y ya no se puede mover en dirección vertical.

La verdad es que queda un poco soso e irreal. Una imagen fija desplazándose por la pantalla... ¡Usemos la grilla y que cambie el aspecto de chuck con el movimiento!

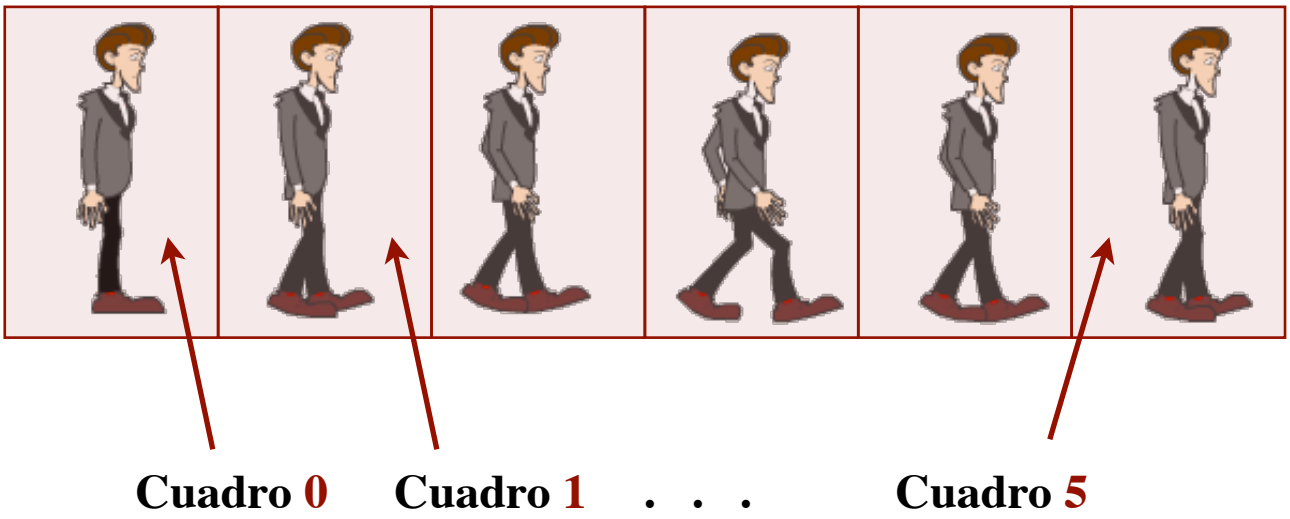
Para eso, hemos de saltar al siguiente paso.

andando03.py

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-
#------------------------------------------------------------------
# andando03.py
# Movimiento simple manual con uso de grilla
#------------------------------------------------------------------

import pilas

pilas.iniciar()

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

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


#Definimos la clase de nuestro actor
class Hombre(pilas.actores.Actor):
    "Un actor que se mueve con las teclas a y s y con animación"

    def __init__(self):
        pilas.actores.Actor.__init__(self)
        self.imagen = pilas.imagenes.cargar_grilla("andando.png", 6)
        self.cuadro = 0
        # Hacemos que el actor se mueva con nuestro control personalizado
        self.aprender(pilas.habilidades.MoverseConElTeclado,
                      control=mandos)

    def actualizar(self):
        # Miramos si se han pulsado las teclas adecudas para cambiar, en
        # su caso, la imagen de la grilla y hacia dónde mira
        if mandos.izquierda:
            if not self.espejado:
                self.espejado = True
            self.cuadro += 1
        elif mandos.derecha:
            if self.espejado:
                self.espejado = False
            self.cuadro += 1
        else:
            self.cuadro = 0
```

```
        if self.cuadro > 5:
            self.cuadro = 1

        self.imagen.definir_cuadro(self.cuadro)

chuck = Hombre()

pilas.ejecutar()
```

La idea es que, en cada paso, el cuadro de animación de la grilla de chuck, cambie. ¡Importante! Recuerda que las listas y demás **elementos iterables**, en **Python**, se numeran empezando por el **0**, no por el 1. Así que los 6 cuadros de nuestra grilla están numerados de la siguiente forma:



Bien. Tenemos que usar una variable que controle el cuadro que está en activo en cada momento, así que la hemos definido como **cuadro** (es decir, **self.cuadro** para que quede almacenada con el Actor) e inicialmente le damos el valor **0**, ya que la posición de partida de chuck es la que está parado.

Y ¿cómo cambiamos de cuadro? ¿Recuerdas que, en cada fotograma de la animación, Pilas llama al método **actualizar()** de cada Actor? ¡Éste es el lugar! Así que añadimos su definición a la clase de actor **Hombre**, para darle un comportamiento personalizado.

Lo primero que hemos de hacer, dentro del método **actualizar()**, es ver si se está pulsando la tecla que mueve a chuck hacia la izquierda o la que lo mueve hacia la derecha o ninguna de las dos. Esto es lo que conseguimos con

```
if mandos.izquierda:
    ...
elif mandos.derecha:
    ...
else:
```

¡Acertaste! Después de definir **mandos** (como un objeto de tipo **pilas.control.Control**), es muy fácil determinar si se ha pulsado la tecla que queremos. Si, por ejemplo, se ha pulsado la tecla que hemos definido como movimiento hacia la izquierda, **mandos.izquierda** es **True** (y **False** en caso contrario). Así que, en el esqueleto anterior, miramos si chuck ha de moverse hacia la izquierda (para hacer lo propio), si no, miramos si hay que moverlo hacia la derecha y, finalmente, en caso contrario (y a través del **else**) hacemos lo correspondiente a cuando chuck tenga que estar parado.

Es evidente que, en cada uno se esos casos, hemos de indicar qué cuadro ha de ser el que empleemos en el dibujado de nuestro personaje. Empezando por el último, lo más sencillo, como el cuadro correspondiente al estado de parado es el número **0**, escribimos

```
        self.cuadro = 0
```

Por lo demás, cuando salgamos del **bloque if** con el cuadro que debemos usar adjudicado, ¿cómo se lo indicamos a Pilas? ¡Ah, amigo! Las **grillas** tienen un método definido para eso. ¡Qué grande es Pilas! :-). Se trata del método **definir_cuadro()**, al que le debemos pasar, como es lógico, el cuadro deseado. Eso es lo que hacemos con

```
        self.imagen.definir_cuadro(self.cuadro)
```

ya que el atributo **self.imagen** del actor contiene la grilla de chuck que hemos cargado en memoria en su **__init__()**. ¿Lo entiendes? ¡Estupendo!

Lo único que nos falta por comprender es lo que hay que hacer cuando se ha pulsado la tecla **a** (es decir, cuando **mandos.izquierda** es **True**) o la tecla **s**. Como ambas funcionan de forma parecida, vamos a fijarnos en el primer caso. Ésta es la parte del código que hemos de analizar:

```
    if mandos.izquierda:
        if not self.espejado:
            self.espejado = True
        self.cuadro += 1
```

Fijando ideas, hay dos cosas que hemos de hacer (recordando que, como chuck ha aprendido la habilidad **pilas.habilidades.MoverseConElTeclado**, ya se desplaza por la pantalla al pulsar las teclas, así que no hemos de preocuparnos de cambiar su coordenada **x**):

- Cambiar el cuadro de la animación.
- Asegurarse de que chuck mira en la dirección en la que está andando.

Lo primero es fácil, basta con aumentar en **1** la variable que controla en qué cuadro de la animación estamos.

Lo segundo cuesta algo más, pero tampoco es complicado. Si te fijas en la grilla de nuestro personaje, verás que en todos los casos está mirando hacia la derecha. Así que la imagen, **en su estado normal** (es decir, cuando el valor del atributo **self.espejado** del actor tiene su valor por defecto, **False**), **mira hacia la derecha**. ¿Qué debemos hacer entonces? Lo que le debemos decir a Pilas es algo como lo siguiente: “Hey, si se ha de mover a la izquierda y la imagen está mirando a la derecha, cámbiala”. Claro, por que si ya está mirando hacia la izquierda, no hace falta, ¿no?...

Manos a la obra. Si chuck está mirando hacia la derecha, **self.espejado** es **False**, así que **not self.espejado** será **True**. Y es en este caso cuando hay que hacer que mire hacia la izquierda, así que hay que decirle que cambie **self.espejado** a **True**. ¡Esto es lo que hacen las dos líneas

```
        if not self.espejado:
            self.espejado = True
```

en el caso de que se halla pulsado la tecla **a** (**mandos.izquierda** es **True**)!

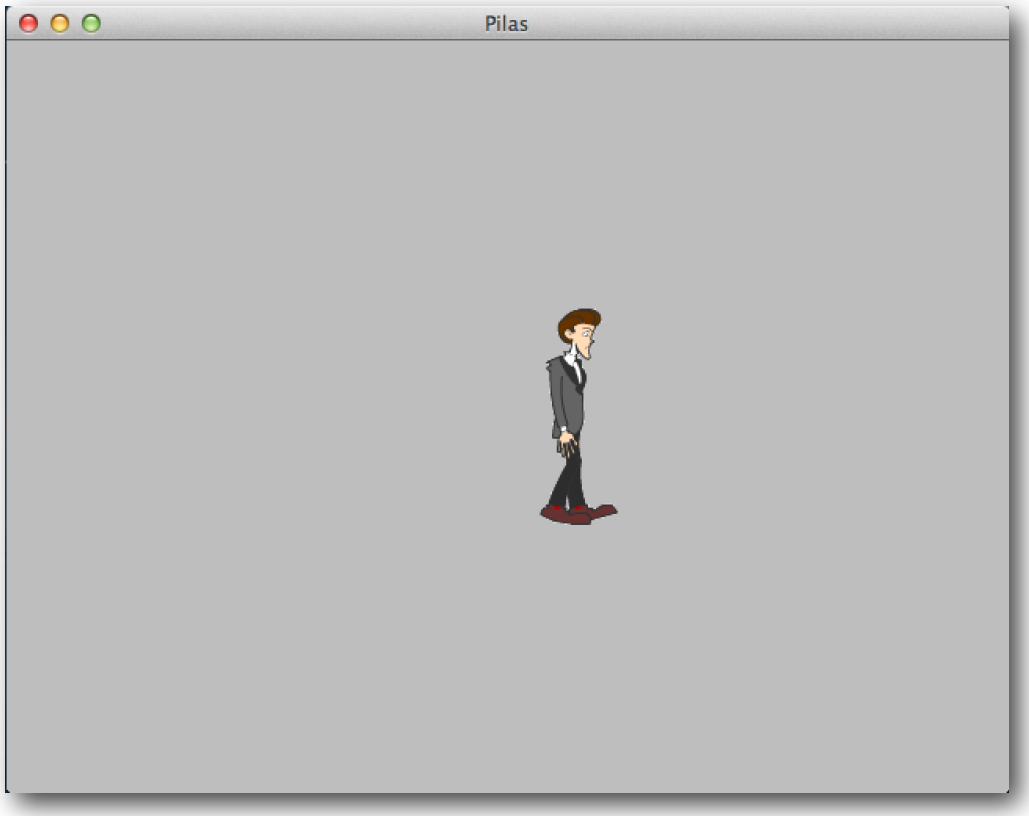
En el caso de que se halla pulsado la tecla **s** y debamos movernos hacia la derecha, ocurre algo parecido pero en sentido inverso, pues hay que asegurarse que **self.espejado** es en ese caso **False**.

Un último matiz. ¡No te olvides que tenemos solo 6 cuadros en la grilla! La parte en la que nuestro personaje anda, empieza en el cuadro número **1** y termina en el cuadro número **5**. Así que cuando sumamos **1** a la variable **self.cuadro**, hemos de asegurarnos de que no nos pasamos de dichos números. Es por eso por lo que hemos añadido

```
        if self.cuadro > 5:
            self.cuadro = 1
```

justo antes de asignar el cuadro que hay que dibujar con el método **definir_cuadro()**.

¡De acuerdo, el movimiento se demuestra andando! Guarda todo el código anterior con el nombre **andando03.py** y ejecútalo. ¿Puedes ver cómo se mueve chuck?



andando04.py

Después de mirar el resultado del paso anterior, sólo podemos llegar a una conclusión: ¡chuck tiene el Baile de San Vito!

Esos movimientos tan convulsivos en los andares de nuestro personaje son consecuencia de la manera grosera con la que hemos abordado los cambios de cuadro de la animación. Para empezar, en cada llamada de Pilas al método **actualizar()** del actor se cambia de cuadro. ¿A qué velocidad, en fotogramas por segundo, ocurre esto? Y ¿cuántos píxeles avanza en cada paso? En una animación correcta, deberíamos controlar estos factores.

Hay diferentes metodologías para abordar esta cuestión. Pero estamos aquí para aprender de Pilas y ésta es una buena excusa para ello...

Más Código Fuente de Pilas

Un paso más avanzado en la compresión del funcionamiento de Pilas lo puedes obtener al examinar su código fuente.

Hasta ahora hemos usado, para hacernos una idea, las funciones `dir()`, `help()` y `pilas.ver()` en el intérprete de Pilas. Pero puedes aprender más, todavía, husmeando entre las líneas de su **código fuente completo**.

Según cual sea la instalación de Pilas que tengas, es posible que no tengas acceso fácil a los archivos `.py` que lo constituyen (y a las imágenes y demás recursos multimedia). Deberían residir en una **carpeta pilas** en tu disco duro.

En cualquier caso, siempre podrás descargarlo desde la web de Pilas

http://www.pilas-engine.com.ar

y, tras descomprimirlo, estudiarlo.

En lo que sigue, vamos a referirnos al código del actor **Pingu**, que puedes encontrar en la **carpeta actores**, dentro de la **carpeta pilas**, con el nombre **pingu.py**. Si lo abres con tu editor (por ejemplo, Ninja-IDE), verás que, a parte del código de la clase **Pingu**, a la que tienes acceso con `pilas.ver()`, tienes el código de otras clases que complementan su definición:

```
class Esperando(Comportamiento):
    "Un actor en posicion normal o esperando a que ..."
    ....
class Caminando(Comportamiento):
    ....
class Saltando(Comportamiento):
    ....
```

Pilas incorpora una serie de actores predefinidos con comportamientos que cubren la mayor parte de las necesidades típicas de los videojuegos. ¿Por qué no aprender de ello? En concreto, tenemos uno de ellos, **Pingu**, que posee una grilla y que avanza de la misma manera que lo hace chuck. Pero, claro, mucho mejor. Ésta es la grilla de Pingu (la puedes encontrar en la **carpeta data** dentro de la **carpeta pilas**, con el nombre **pingu.png**):

Si abres el intérprete de Pilas y trasteas con él (crea una instancia de este tipo de actor escribiendo **pilas.actores.Pingu()**), verás que se desplaza de una forma similar a lo que deseamos (con la diferencia de que no usa el espejado y salta). De los cuadros que forman su grilla, la posición 'en reposo' es la número **4** y el caminar se forma con los cuadros del **5** al **9**.

¿Aprendemos de su código? Su método **__init__()** es el siguiente:

```
def __init__(self, x=0, y=0):
    Actor.__init__(self, x=x, y=y)
    self.imagen = pilas.imagenes.cargar_grilla("pingu.png", 10)
    self.definir_cuadro(4)
    self.hacer(Esperando())
    self.radio_de_colision = 30
    self.centro = ("centro", "abajo")
```

No hace falta que lo comprendamos todo al cien por cien. Fíjate sólo en un par de cosas. Para empezar, que la grilla consta, como sabemos, de **10** cuadros y que se adjudica por defecto el cuadro **4** (como también sabemos que debe ser). La forma de hacer esto último tiene su matiz, pues en lugar de acceder directamente al método **definir_cuadro()** de la grilla, se usa un método definido en el propio **Pingu**, más adelante, para que sea más accesible desde otros puntos del programa: en lugar de referirse a él como **imagen.definir_cuadro()** podemos poner, directamente, **definir_cuadro()**. Es una cuestión de estilo.

A lo que vamos. Esto es lo fundamental: **se usa el concepto de Comportamiento para definir qué debe hacer Pingu**. ¿Qué comportamiento le adjudicamos? Éste:

```
    self.hacer(Esperando())
```

¿Entiendes el proceso? Como puedes comprobar usando **help()**, el método **hacer()** de un actor define el comportamiento de manera inmediata. Así que el quid de la cuestión está en cómo se define el comportamiento que hemos denominado **Esperando**. Ya hemos visto, en el recuadro de más arriba, que su definición está en el mismo archivo **pingu.py**.

De la misma manera que un actor genérico reside en el módulo **pilas.actores** y usamos **pilas.actores.Actor** como el padre del cual derivar los demás, los comportamientos residen en el módulo **pilas.comportamientos** y el progenitor de todos ellos es

pilas.comportamientos.Comportamiento. Eso sí, por abreviar el código, en el archivo **pingu.py** se importan las librerías de manera que podamos usar directamente sus nombres... Éste es el código de la clase de comportamiento **Esperando** de Pingu:

```
class Esperando(Comportamiento):
    "Un actor en posicion normal o esperando a que el usuario pulse alguna tecla."

    def iniciar(self, receptor):
        self.receptor = receptor
        self.receptor.definir_cuadro(4)

    def actualizar(self):
        if pilas.escena_actual().control.izquierda:
            self.receptor.hacer(Caminando())
        elif pilas.escena_actual().control.derecha:
            self.receptor.hacer(Caminando())

        if pilas.escena_actual().control.arriba:
            self.receptor.hacer(Saltando())
```

Pilas llama, tras crear un comportamiento, a su método **iniciar()** (algo similar al método **__init__()** que conocemos) pasándole como argumento (**receptor**) el objeto al que se le va a asociar. En nuestro caso, **receptor** se refiere a **Pingu** y podemos ver cómo se insiste en indicar que el cuadro por defecto es el **4**.

¿Qué encontramos en el familiar método **actualizar()** que, recordemos, se llama en cada fotograma de la animación de Pilas? Debería ya ser fácil de entender: se mira el control del teclado y, si se ha pulsado una tecla de movimiento, se procede al comportamiento correspondiente. Nos interesa el comportamiento **Caminando**. Su definición está, de nuevo, en el mismo archivo **pingu.py** más adelante:

```
class Caminando(Comportamiento):

    def __init__(self):
        self.cuadros = [5, 5, 6, 6, 7, 7, 8, 8, 9, 9]
        self.paso = 0

    def actualizar(self):
        self.avanzar_animacion()

    ...
```

Sólo hemos puesto las primeras líneas del código del comportamiento **Caminando**, pues ya entraremos en detalles en seguida, cuando pasemos a nuestro chuck. Basta que veamos que se indica la lista de cuadros que formarán parte de la animación del comportamiento (¿recuerdas la grilla de **Pingu**?). Observa que aparecen por duplicado; eso hará que se muestre más lenta la transición al tener que pasar por el doble de cuadros.

En resumen de todo lo que hemos visto: para conseguir que nuestro personaje responda de un modo específico, **tenemos que definir comportamientos y aplicárselos con el método hacer()**.

Ya estamos en condiciones de atacar nuestra mejora para chuck. ¡Aquí va!

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-
#---------------------------------------------------------------------
# andando04.py
# Movimiento animado personalizado usando grilla
#---------------------------------------------------------------------

import pilas

pilas.iniciar()

VELOCIDAD = 4

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

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


#Definimos la clase de nuestro actor
class Hombre(pilas.actores.Actor):
    "Un actor que se mueve con las teclas a y s y con animación"

    def __init__(self):
        pilas.actores.Actor.__init__(self)
        self.imagen = pilas.imagenes.cargar_grilla("andando.png", 6)
        self.definir_cuadro(0)
        # Hacemos que el actor se mueva con el comportamiento personalizado
        self.hacer(Esperando())

    def definir_cuadro(self, indice):
        self.imagen.definir_cuadro(indice)


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
        self.receptor.definir_cuadro(0)

    def actualizar(self):
        if mandos.izquierda:
            self.receptor.hacer(Caminando())
        elif mandos.derecha:
```

```
            self.receptor.hacer(Caminando())


class Caminando(pilas.comportamientos.Comportamiento):
    "Actor caminando"

    def iniciar(self, receptor):
        self.receptor = receptor
        self.cuadros = [1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 5, 5, 5, 5]
        self.paso = 0

    def actualizar(self):
        self.avanzar_animacion()

        if mandos.izquierda:
            if not self.receptor.espejado:
                self.receptor.espejado = True
            self.receptor.x -= VELOCIDAD
        elif mandos.derecha:
            if self.receptor.espejado:
                self.receptor.espejado = False
            self.receptor.x += VELOCIDAD
        else:
            self.receptor.hacer(Esperando())

    def avanzar_animacion(self):
        self.paso += 1
        if self.paso >= len(self.cuadros):
            self.paso = 0

        self.receptor.definir_cuadro(self.cuadros[self.paso])


chuck = Hombre()

pilas.ejecutar()
```

En primer lugar, definimos la constante **VELOCIDAD** que usaremos más adelante para controlar cuántos píxeles avanzará chuck en cada paso. Recuerda el convenio que usamos para los nombres; poner las constantes en mayúsculas nos sirve para identificarlas rápidamente.

Pasando a la definición de la clase del actor, nos encontramos con la función

```
    def definir_cuadro(self, indice):
        self.imagen.definir_cuadro(indice)
```

que, como hemos dicho antes, nos permite cambiar el cuadro de la animación de chuck sin necesidad de acceder directamente a la grilla. Esto es precisamente lo que hacemos en el método **__init__()**, indicando que el cuadro **0** es el cuadro en reposo por defecto. Y, a continuación, es donde establecemos también el comportamiento inicial del personaje; nuestro conocido **Esperando**.

Pasemos al quid de la cuestión; la implementación de los comportamientos de chuck. Tenemos dos; chuck parado (que se corresponde con **Esperando**) y chuck caminando (que se corresponderá con **Caminando**). Hay que tener en cuenta que en cualquier momento se puede pasar de un comportamiento al otro (basta con que el jugador pulse o deje de pulsar la tecla apropiada), así que debemos implementarlo en el código. Éste es el de **Esperando**:

```
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
        self.receptor.definir_cuadro(0)

    def actualizar(self):
        if mandos.izquierda:
            self.receptor.hacer(Caminando())
        elif mandos.derecha:
            self.receptor.hacer(Caminando())
```

¿Te resulta familiar la estructura? ¿La comprendes? En el método **iniciar()** del comportamiento recogemos la referencia al personaje al que se lo vamos a aplicar en la variable **receptor** y ponemos su cuadro a **0**, como ya sabemos. Nada más. Y en el método **actualizar()** nos aseguramos de que si se pulsa la tecla **a** o la tecla **s**, se cambie el comportamiento a **Caminando**.

¡No es tan difícil! ¿Qué hay del código de Caminando? Veamos:

```
class Caminando(pilas.comportamientos.Comportamiento):
    "Actor caminando"

    def iniciar(self, receptor):
        self.receptor = receptor
        self.cuadros = [1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 4, 5, 5, 5]
        self.paso = 0

    def actualizar(self):
        self.avanzar_animacion()

        if mandos.izquierda:
            if not self.receptor.espejado:
                self.receptor.espejado = True
            self.receptor.x -= VELOCIDAD
        elif mandos.derecha:
            if self.receptor.espejado:
                self.receptor.espejado = False
```

```
            self.receptor.x += VELOCIDAD
        else:
            self.receptor.hacer(Esperando())

    def avanzar_animacion(self):
        self.paso += 1
        if self.paso >= len(self.cuadros):
            self.paso = 0

        self.receptor.definir_cuadro(self.cuadros[self.paso])
```

Éste es un poco más largo, pero si lo desgranamos deberías comprenderlo bien.

Notarás que en el método **iniciar()** definimos dos variables; una con la lista de cuadros que forman la animación, **self.cuadros**, y otra que llevará la cuenta de cuál es el cuadro que está en activo en cada momento, **self.paso** (a **0** inicialmente, que indica el primer cuadro de la lista, es decir, el cuadro **1**). Si somos unos verdaderos artistas, tendremos mucho detalle en los diferentes fotogramas que forman la animación de un actor y parecerá un movimiento extremadamente real. Como no es el caso y hemos usado un sprite que hemos descargado directamente de internet, hemos triplicado la referencia a cada cuadro para que ésta se muestre más lenta y progresiva.

El método que se encarga de avanzar los cuadros en la animación, no podía llamarse de otro modo, es **avanzar_animacion()**. Fíjate que lo que hace es muy transparente: Añade **1** a la variable **paso**, si ésta es mayor que el número de cuadros que hay en la lista (lo que se averigua con la función **len()**) se resetea a **0** (es decir, el primer cuadro de la lista) y se invica al método **definir_cuadro()** del personaje indicando a cuál hay que cambiar.

Bien. Esto nos deja en el método **actualizar()** que es donde debemos mirar qué tecla se está pulsando para hacer una cosa u otra. Observa su código...

Lo primero, no en vano estamos caminando, es avanzar un cuadro en la animación del movimiento usando el método que acabamos de ver, **avanzar_animacion()**. Lo siguiente es mirar el control del teclado que hemos definido, **mandos**, y ver si pulsamos la tecla **a** o la tecla **s**. En caso contrario, fíjate en el **else**, se cambia el comportamiento a **Esperando**. ¡Claro!

Supongamos, por ejemplo, que se ha pulsado la tecla **a** para desplazar a chuck hacia la izquierda. Aquí hacemos algo similar a lo que hicimos en el paso anterior con el espejado del actor, ¿recuerdas? Pero además, hemos de avanzar a chuck para que no se quede en el sitio, de ahí que restemos la constante **VELOCIDAD** a su coordenada x. Y en el caso de que se pulse la tecla **s**, hemos de hacer lo mismo pero aumentando su coordenada x.

¡Y ya está todo! ¿Ves cómo no era tan complicado? Pruébalo tu mismo. Guarda todo el código con el nombre **andando04.py** y ejecútalo. Notarás que la animación es mucho más suave. Y sólo cambiando el valor que damos a **VELOCIDAD** y/o la lista de cuadros, podemos controlar mucho mejor el comportamiento de chuck.

¡Pero eso no es todo! Ya nos hemos envalentonado... Si el Pingu original saltaba, ¿por qué no puede hacerlo también nuestro chuck? De paso, aprendemos como añadir más características a nuestros comportamientos... en el próximo apartado.

andando05.py

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-
#---------------------------------------------------------------------
# andando05.py
# Andando y saltando usando grilla
#---------------------------------------------------------------------

import pilas

pilas.iniciar()

VELOCIDAD = 4

# Definimos las teclas que moverán al personaje
teclas = {pilas.simbolos.a: 'izquierda', pilas.simbolos.s: 'derecha',
          pilas.simbolos.ESPACIO: 'arriba'}

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


#Definimos la clase de nuestro actor
class Hombre(pilas.actores.Actor):
    "Un actor que se mueve con las teclas a, s y ESPACIO y con animación"

    def __init__(self):
        pilas.actores.Actor.__init__(self)
        self.imagen = pilas.imagenes.cargar_grilla("andando.png", 6)
        self.definir_cuadro(0)
        # Hacemos que el actor se mueva con el comportamiento personalizado
        self.hacer(Esperando())

    def definir_cuadro(self, indice):
        self.imagen.definir_cuadro(indice)


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
        self.receptor.definir_cuadro(0)
```

```
    def actualizar(self):
        if mandos.izquierda:
            self.receptor.hacer(Caminando())
        elif mandos.derecha:
            self.receptor.hacer(Caminando())
        if mandos.arriba:
            self.receptor.hacer(Saltando())


class Caminando(pilas.comportamientos.Comportamiento):
    "Actor caminando"

    def iniciar(self, receptor):
        self.receptor = receptor
        self.cuadros = [1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 5, 5, 5, 5]
        self.paso = 0

    def actualizar(self):
        self.avanzar_animacion()

        if mandos.izquierda:
            if not self.receptor.espejado:
                self.receptor.espejado = True
            self.receptor.x -= VELOCIDAD
        elif mandos.derecha:
            if self.receptor.espejado:
                self.receptor.espejado = False
            self.receptor.x += VELOCIDAD
        else:
            self.receptor.hacer(Esperando())
        if mandos.arriba:
            self.receptor.hacer(Saltando())

    def avanzar_animacion(self):
        self.paso += 1
        if self.paso >= len(self.cuadros):
            self.paso = 0

        self.receptor.definir_cuadro(self.cuadros[self.paso])


class Saltando(pilas.comportamientos.Comportamiento):
    "Actor Saltando"

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

```
        self.receptor.definir_cuadro(0)
        self.origen = self.receptor.y
        self.dy = 10

    def actualizar(self):
        self.receptor.y += self.dy
        self.dy -= 0.3

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

        if mandos.izquierda:
            self.receptor.x -= VELOCIDAD
        elif mandos.derecha:
            self.receptor.x += VELOCIDAD


chuck = Hombre()

pilas.ejecutar()
```

Bueno, manos a la obra. Lo primero que necesitamos es elegir una tecla para que cuando la pulsemos hagamos que nuestro personaje salte. Hemos elegido la tecla **ESPACIO** y, por lo tanto, hemos modificado el diccionario **teclas** para ello.

¿Qué más hemos de hacer? Por supuesto; tenemos que definir un comportamiento para el salto, al que denominaremos (muy originalmente) **Saltando**. Antes de ver cómo hemos implementado su código, fíjate que tenemos que contemplar esa posibilidad en los demás comportamientos, ya que la tecla que lo activa puede pulsarse en cualquier momento. Dicho y hecho. Lo verás en el **if** con **mandos.arriba**, tanto en **Esperando** como en **Caminando**.

Ya solo queda comprender el código del comportamiento **Saltando**. ¿Cómo podemos hacer que un personaje suba y luego baje? Y no sólo, eso; ¿cómo podemos hacer que baje solo hasta la misma posición desde la que ha partido? La respuesta a esto último se puede adivinar enseguida; almacenando la coordenada **y** de chuck cuando empieza a saltar y deteniendo su movimiento vertical cuando vuelva a alcanzarse ese valor. Y el movimiento vertical lo podemos hacer sumando una cierta cantidad a la coordenada **y** en cada fotograma de la animación. Ese es el sentido de las dos variables definidas en el método **iniciar()**:

```
        self.origen = self.receptor.y
        self.dy = 10
```

En efecto; **origen** almacena la coordenada **y** de la que parte el salto y **dy** lo usaremos para sumarlo a dicha coordenada **y**, empezando por **10** píxeles. Fíjate, de paso, que en el salto usamos como imagen el cuadro **0**, a falta de otro más adecuado.

El resto está en el método **actualizar()**: cada vez que se llama, se suma la cantidad **dy**, y ésta se disminuye un poco (en **0.3** unidades) para hacer que suba cada vez menos rápidamente, simulando la gravedad. Variando ese valor de **0.3**, puedes conseguir una gravedad aparente más o menos intensa. Observa también el siguiente **if**:

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

¡Hace exactamente lo que hemos comentado dos párrafos más arriba! En cuanto chuck bajara más de la altura desde la que ha empezado el salto, su coordenada **y** se vuelve a dejar fija y se cambia el comportamiento a **Esperando**, abandonando el salto.

Y una última cosa; no hemos de olvidar que durante el salto, nuestro personaje avanza también hacia la derecha o hacia la izquierda; de eso se encargan las últimas líneas del método **actualizar()**, ya conocidas.

¡Perfecto! Guarda todo el código anterior con el nombre **andando05.py** y ejecútalo... ¿No es bonito ver cómo lo hemos ido educando poco a poco? Ah, es como un hijo... :-)

Vamos a dejarlo aquí. Te podemos sugerir muchas mejoras para que experimentes. Por ejemplo, durante el salto, chuck se empeña en mirar siempre hacia el mismo lado, pulsemos la tecla que pulsemos. ¿Qué tal arreglar eso? O, burlando las leyes de la física, cuando está en el aire, es capaz de cambiar de dirección... ¿Sabrías hacerlo más real? Y ¿qué te parecería hacer que salte cuando se haga click con el ratón? ¿Y usar tu propio personaje, añadir un fondo...? El mar de la creatividad se extiende ante nosotros...