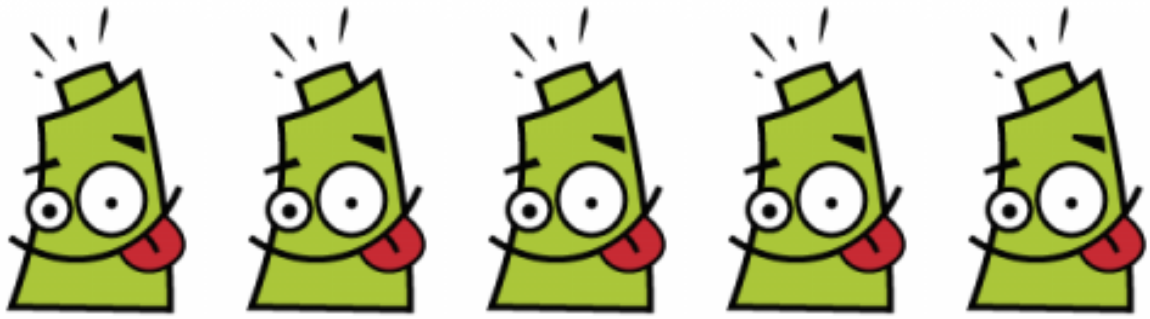
Ingeniería en Sistemas de Información
Ingeniería de Software de Fuentes Abiertas/Libres



Comunidad Pilas Engine

ANEXO - MANUAL

Zahradnicek Desirée

Ingeniería de Software de fuentes Abiertas/Libres

Participación en Comunidad

Cátedra:

- Ingeniería de Software de fuentes Abiertas/Libres

Docente:

- Medel, Ricardo Hugo

Alumna:

- Zahradnicek, Desirée Legajo: 60003

UNIVERSIDAD TECNOLÓGICA NACIONAL

FACULTAD REGIONAL CÓRDOBA

- 2017 -

En este tutorial crearemos el juego Torres de Hanói, el cual consiste en mover toda la torre desde el poste inicial hacia cualquiera de los otros dos postes, con las siguientes reglas:

- Sólo se puede mover de a un disco por vez
- No se pueden mover discos que tengan otro disco encima
- No se puede colocar un disco mayor encima de otro

Para comenzar vamos a ver cómo utilizar los métodos que tiene pilas para dibujar, en particular rectángulos, y así crear los discos y postes sin necesidad de recurrir a imágenes externas. Lo primero que hace falta es crear una superficie sobre la cuál realizaremos el dibujo, para esto llamamos al método:

pilas.imagenes.cargar_superficie(ancho,alto), el cual nos devuelve el objeto <Superficie>.

```
superficie = pilas.imagenes.cargar_superficie(ancho,alto)
```

Luego sobre esta superficie podemos utilizar distintos métodos para dibujar sobre ella, en este caso utilizaremos el método rectángulo(x, y, ancho, alto, color=colores.negro, relleno=False, grosor=1) el cuál dibuja un rectángulo desde el punto (x,y) hasta el punto (x+ancho,y+alto). Si el argumento relleno es True el rectángulo será lleno, sino sólo dibujará el borde.

```
superficie.rectangulo(0,0,ancho,alto,color=pilas.colores.blanco,relleno=True)
```

Finalmente, para que aparezca en pantalla lo que dibujamos, debemos crear un actor y decirle que utilice la superficie que creamos como imagen, de la siguiente forma:

```
actor = pilas.actores.Actor(imagen=superficie)
```

A continuación, se muestra el código completo hasta ahora:

```
# coding: utf-8
```

```
import pilasengine

pilas = pilasengine.iniciar()

ancho = 100
alto = 20

superficie = pilas.imagenes.cargar_superficie(ancho,alto)
superficie.rectangulo(0,0,ancho,alto,color=pilas.colores.blanco,relleno=True)
actor = pilas.actores.Actor(imagen=superficie)

actor.aprender(pilas.habilidades.Arrastrable)

pilas.ejecutar()
```

En el código anterior agregamos actor.aprender(pilas.habilidades.Arrastrable) para hacer que el rectángulo sea arrastrable, sólo porque nos será útil mientras estemos desarrollando el juego.

Como vamos a necesitar crear muchos rectángulos para el juego, uno por cada disco, uno por cada poste y uno de base; es útil crear una función a la cuál le podamos pasar los parámetros necesarios y nos devuelva ya el actor creado con un rectángulo como imagen, definimos entonces la función crear_rectangulo de la siguiente manera:

```
def crear_rectangulo( ancho, alto, color=pilas.colores.blanco, texto="",
colortexto=pilas.colores.negro ):
    superficie = pilas.imagenes.cargar_superficie(ancho,alto)
    superficie.rectangulo(0,0,ancho,alto,color=color, relleno=True)
    superficie.rectangulo(0,0,ancho,alto,color=pilas.colores.negro, relleno=False,grosor=4)
    superficie.texto(texto, magnitud=12, color=colortexto,x=ancho/2-5)
    actor = pilas.actores.Actor(imagen=superficie)
    actor.aprender(pilas.habilidades.Arrastrable)
    return actor
```

En esta función además de dibujar un rectángulo lleno del color deseado, también se dibuja un rectángulo sin relleno con bordes de color negro, y se escribe un texto que comienza aproximadamente en el centro del rectángulo, el cuál utilizaremos para numerar los discos y postes. Para no cometer errores en las posiciones de los postes, discos y base se definen a continuación los siguientes parámetros que utilizaremos:

```
# Número de discos del juego
numero_discos = 5

# Parámetros gráficos del juego
ancho_max = 200
ancho_min = 50
alto = 20
separacion_postes = 200
alto_postes = alto*(numero_discos+1)
```

Con esto podríamos crear el primer poste de la siguiente forma:

```
poste0 =
crear_rectangulo(ancho_postes,alto_postes,color=pilas.colores.negro,texto=str(0),colortexto=
pilas.colores.blanco)
```

pero, para crear los tres postes de forma más sencilla, y tener todos los objetos en una lista podemos hacer lo siguiente:

```
postes = [
crear_rectangulo(ancho_postes,alto_postes,color=pilas.colores.negro,texto=str(i),colortexto=p
ilas.colores.blanco) for i in range(3) ]
```

de esta forma postes es una lista, que contiene el poste número 0 en la posición 0, el poste 1 en la posición 1 y el poste 2 en la posición 2.

Hasta ahora el código que tenemos es el siguiente:

```
# coding: utf-8
```

```
import pilasengine

pilas = pilasengine.iniciar()

# Número de discos del juego
numero_discos = 5

# Parámetros gráficos del juego
ancho_max = 200 # Ancho del disco más grande
ancho_min = 50  # Ancho del disco más chico
alto = 20       # Alto de los discos y la base
separacion_postes = 200             # Separación entre los postes
alto_postes = alto*(numero_discos+1) # Altura de los postes
ancho_postes = 14                   # Ancho de los postes

def crear_rectangulo( ancho, alto, color=pilas.colores.blanco, texto="",
colortexto=pilas.colores.negro ):
    superficie = pilas.imagenes.cargar_superficie(ancho,alto)
    superficie.rectangulo(0,0,ancho,alto,color=color, relleno=True)
    superficie.rectangulo(0,0,ancho,alto,color=pilas.colores.negro, relleno=False,grosor=4)
    superficie.texto(texto, magnitud=12, color=colortexto,x=ancho/2-5)
    actor = pilas.actores.Actor(imagen=superficie)
    actor.aprender(pilas.habilidades.Arrastrable)
    return actor

postes = [
crear_rectangulo(ancho_postes,alto_postes,color=pilas.colores.negro,texto=str(i),colortexto=p
ilas.colores.blanco) for i in range(3) ]

pilas.ejecutar()
```

Al ejecutar el código vemos que todos los postes se encuentran en el centro de la pantalla, uno sobre el otro. Esto es así ya que solamente los hemos creado, todavía no les dimos una posición. Para esto vamos a cambiar las coordenadas X, Y de cada poste, puede probar modificar el valor en el intérprete de la siguiente manera:

```
poste[0].x = -200
```

o bien así:

```
poste[2].x = [+200]
```

Como puede verse la segunda forma de definir la posición mediante una lista hace que el poste 2 se mueva de forma continua hasta la posición deseada. Ésta es la forma más sencilla que tiene pilas para animar a los actores, pruebe que sucede si en la lista usted pone más de un valor (poste[2].x = [+100,-100,+200]). Entonces vamos a ubicar los postes y crear la base de la siguiente forma:

```
for i in range(3):
    postes[i].x = separacion_postes*(i-1)
    postes[i].y = alto_postes/2-alto/2
base = crear_rectangulo(2*separacion_postes+ancho_max,alto,pilas.colores.negro)
base.y = -alto
```

Los discos los vamos a crear de forma muy similar a los postes, y los vamos a guardar a todos en una lista llamada discos, la principal diferencia que tenemos ahora es que todos los discos deben ser de distinto tamaño, desde ancho_max hasta ancho_min y debe haber exactamente numero_discos. Para lograr esto vamos a necesitar calcular la diferencia de ancho que debe haber entre cada disco, esto es:

```
delta = ( ancho_max - ancho_min )/( numero_discos - 1 )
```

Para entender el porqué del -1 la forma más sencilla es pensar en el caso con menos discos posibles, es decir 2, y en este caso la diferencia de ancho entre el más chico y el más grande debe ser exactamente ancho_max - ancho_min. Hecho este cálculo auxiliar, entonces podemos crear los discos de la siguiente manera:

```
    discos = [ crear_rectangulo(ancho_max-delta*i,alto,texto=str(i)) for i in
range(numero_discos)]
```

Luego vamos a ubicar todos los discos en el primer poste, por lo que el posicionamiento de los discos se puede hacer así:

```
    for i in range(numero_discos):
        discos[i].y = alto*i
        discos[i].x = -separacion_postes
```

Hasta ahora el código que tenemos es el siguiente:

```
    # coding: utf-8
    import pilasengine

    pilas = pilasengine.iniciar()

    # Número de discos del juego
    numero_discos = 5

    # Parámetros gráficos del juego
    ancho_max = 200
    ancho_min = 50
    alto = 20
    separacion_postes = 200
    alto_postes = alto*(numero_discos+1)
    ancho_postes = 14

    def crear_rectangulo( ancho, alto, color=pilas.colores.blanco, texto="",
colortexto=pilas.colores.negro ):
        superficie = pilas.imagenes.cargar_superficie(ancho,alto)
        superficie.rectangulo(0,0,ancho,alto,color=color, relleno=True)
```

```
        superficie.rectangulo(0,0,ancho,alto,color=pilas.colores.negro, relleno=False,grosor=4)
        superficie.texto(texto, magnitud=12, color=colortexto,x=ancho/2-5)
        actor = pilas.actores.Actor(imagen=superficie)
        actor.aprender(pilas.habilidades.Arrastrable)
        return actor

    # Crear los postes y base
    postes = [
crear_rectangulo(ancho_postes,alto_postes,color=pilas.colores.negro,texto=str(i),colortexto=p
ilas.colores.blanco) for i in range(3) ]
    for i in range(3):
        postes[i].x = separacion_postes*(i-1)
        postes[i].y = alto_postes/2-alto/2
    base = crear_rectangulo(2*separacion_postes+ancho_max,alto,pilas.colores.negro)
    base.y = -alto

    # Crear los discos
    delta = (ancho_max - ancho_min)/(numero_discos-1)
    discos = [ crear_rectangulo(ancho_max-delta*i,alto,texto=str(i)) for i in
range(numero_discos)]
    for i in range(numero_discos):
        discos[i].y = alto*i
        discos[i].x = -separacion_postes

    pilas.ejecutar()
```

Como podemos ver el código crea los postes, la base y los discos y los ubica correctamente en la posición inicial, incluso si cambiamos numero_discos por otros valores. Ahora que ya tenemos la base para poder ver el juego, es momento de centrarnos en la lógica del mismo.

Como sucede con la mayoría de juegos, es necesario tener alguna forma de representar el estado en el que éste se encuentra. En este caso una forma sencilla de representar el estado es mediante listas, podemos tener una lista por cada poste, que contenga el número de disco que se encuentra en él. Por ejemplo, si en un poste se encuentran los discos 0,1,3 y 4 de abajo hacia

arriba la lista que representaría esto sería [0,1,3,4]. Teniendo en cuenta que tenemos tres postes, entonces podemos tener una lista. Por ejemplo, si en el poste 0 se encuentran los discos 0,1,3; en el poste 1 los discos 2,4 y en el poste 2 no hay ningún disco el estado sería el siguiente [[0,1,3], [2,4], []].

Es decir, tenemos una lista de listas de discos. El estado inicial sería [[0,1,2,3,...,numero_discos-1] , [] , []], y esto puede programarse así:

```
# Estado inicial del juego
juego = [range(numero_discos),[],[]]
```

También vamos a necesitar alguna forma de describir un movimiento a realizar. Como el único disco que puede moverse es el que se encuentra arriba, y si es válido el movimiento se lo coloca también en la parte superior de otro poste, lo único que hace falta para describir un movimiento es el poste desde el cuál sacar un disco (origen) y el poste sobre el cuál colocarlo (destino).

Con esto en cuenta, vamos entonces a crear una función validar_movimiento(origen,destino) que devuelva True si el movimiento es válido de acuerdo a las reglas del juego Torres de Hanói, y False en caso contrario. Primero resolvamos lo trivial, si el poste origen y poste destino son iguales, en realidad no se estaría realizando ningún movimiento, así que ya podemos devolver False, hasta ahora tendríamos lo siguiente:

```
def validar_movimiento(origen,destino):
    if origen == destino:
        return False
```

Sigamos con lo sencillo, si el poste origen no tiene ningún disco, es imposible realizar el movimiento, por lo que nuevamente podemos devolver False:

```
def validar_movimiento(origen,destino):
    if origen == destino:
        return False
    if len(juego[origen]) > 0:
        # Hay al menos un disco en poste origen, tenemos que seguir verificando
    else:
        return False
```

Ahora, si en poste origen hay algún disco es posible realizar un movimiento, y entonces nos quedan dos posibilidades, que en poste destino haya o no algún disco. El caso más sencillo es el que en poste destino no hay ningún disco, por lo que el movimiento es válido sin importar el disco que se quiera mover. Hasta ahora tenemos lo siguiente:

```
def validar_movimiento(origen,destino):
    if origen == destino:
        return False
    if len(juego[origen]) > 0:
        if len(juego[destino]) == 0:
            return True
        else:
            # Tenemos que verificar que los tamaños de discos sean compatibles
    else:
        return False
```

Por último, nos queda verificar el caso más complejo, que es cuando se quiere mover un disco a un poste con discos, donde debemos verificar que el disco que se quiere mover sea más chico que el disco superior de poste destino. Acceder al último elemento en una lista en Python es muy sencillo, simplemente se debe usar un índice -1 como en una_lista[-1]. Recordando que mientras más chico es un disco más alto es el número que lo representa, la comparación que se debe hacer es la siguiente:

```
juego[origen][-1] > juego[destino][-1]
```

Explicación: juego es una lista que contiene tres listas, una por cada poste, donde cada una de éstas contiene un número que representa a cada disco. Entonces juego[origen] es la lista que corresponde al poste origen, y juego[destino] es la lista que corresponde al poste destino. Al hacer juego[origen][-1] se está accediendo al último elemento de la lista que representa a los discos en el poste origen, el que según nuestra forma de representar el estado se encuentra más arriba; y similarmente al hacer juego[destino][-1]. Con esto la función validar_movimiento queda completa:

```
def validar_movimiento(origen,destino):
    if origen == destino:
```

```
        return False
    if len(juego[origen]) > 0:
        if len(juego[destino]) == 0:
            return True
        else:
            return juego[origen][-1] > juego[destino][-1]
    else:
        return False
```

Para verificar que la función validar_movimiento funciona correctamente, en el intérprete usted puede asignar el valor de juego por ejemplo a [[0,1,2],[3,4],[]] y probar todos los posibles movimientos de la siguiente forma y ver que los resultados sean correctos:

```
for origen in range(3):
    for destino in range(3):
        print(origen,destino,validar_movimiento(origen,destino))
```

Ahora empezaremos a darle forma a la función más importante, realizar_movimiento(origen,destino), la cual se encargará como su nombre lo indica de realizar el movimiento deseado de ser posible. Utilizando la función validar_movimiento creada la estructura de esta función es la siguiente:

```
def realizar_movimiento(origen,destino):
    if validar_movimiento(origen,destino):

        # realizar el movimiento
        pilas.avisar(str(juego))

    else:
        pilas.avisar(u"Movimiento inválido")
```

Esta función debe realizar dos tareas distintas, en primer lugar, debe actualizar la variable juego de acuerdo al movimiento deseado, y en segundo lugar debe realizar gráficamente el

movimiento. Vamos a resolver ahora como actualizar el estado. Para esto, antes agregaremos las siguientes líneas al final de la función para que nos permitan visualizar el estado del juego:

```
pilas.avisar(str(juego))
print(juego)
```

Para realizar el movimiento en sí, debemos primero sacar el último elemento de la lista que representa al poste origen, esto podemos hacerlo utilizando el método pop() de la clase list en Python, el cuál elimina el último elemento de la lista y lo retorna, de la siguiente manera:

```
disco_a_mover = juego[origen].pop()
```

Luego debemos agregar este valor a la lista del poste destino, para lo que podemos usar el método append(elemento) también de la clase list de Python el cuál agrega el elemento pasado al final de la lista, de la siguiente forma:

```
juego[destino].append(disco_a_mover)
```

Con esto la función realizar_movimiento que tenemos hasta ahora es la siguiente:

```
def realizar_movimiento(origen,destino):
    if validar_movimiento(origen,destino):

        # actualizar el estado del juego
        disco_a_mover = juego[origen].pop()
        juego[destino].append(disco_a_mover)


        pilas.avisar(str(juego))


    else:
        pilas.avisar(u"Movimiento inválido")
    print(juego)
```

Puedes probar en el intérprete llamar a la función realizar_movimiento(origen,destino) y ver como modifica el vector de estado del juego.

Ahora vamos a ver cómo resolver gráficamente el movimiento, el cuál vamos a realizar antes de actualizar el estado del juego, por lo que juego[origen][-1] es efectivamente el número del disco que vamos a mover. Para esto, debemos resolver la posición final que debe tomar el disco, la posición en X es sencilla, ya que sólo depende del poste destino al cuál queremos moverlo, pero la posición en Y, depende de la cantidad de discos que haya en el poste destino. El cálculo que puede razonar es el siguiente:

```
x_final = separacion_postes*(destino-1)
y_final = alto*len(juego[destino])
```

Con esto actualizamos la función realizar movimiento de la siguiente manera:

```
def realizar_movimiento(origen,destino):
    if validar_movimiento(origen,destino):

        # realizar movimiento
        x_final = separacion_postes*(destino-1)
        y_final = alto*len(juego[destino])
        disco_a_mover = juego[origen][-1]
        discos[disco_a_mover].x = x_final
        discos[disco_a_mover].y = y_final

        # actualizar el estado del juego
        disco_a_mover = juego[origen].pop()
        juego[destino].append(disco_a_mover)

        pilas.avisar(str(juego))

    else:
        pilas.avisar(u"Movimiento inválido")
```

```
    print(juego)
```

Como se puede ver, al llamar a la función realizar_movimiento(origen,destino) se mueven los discos, pero estos lo hacen de manera inmediata. ¿No sería mucho más agradable que el disco se moviera de forma suave desde un poste al otro? Claro que sí, y no es mucho más difícil de realizar que lo que tenemos hasta ahora. Vamos a realizar ahora la animación, haciendo que primero el disco suba por encima del poste, luego se mueva horizontalmente hasta el poste destino, y finalmente baje hasta apoyar sobre otro disco o la base. Para esto necesitamos dos cálculos auxiliares más, la posición X inicial del movimiento y la altura a la cual se debe elevar el disco; los cuales resultan en:

```
x_inicial = separacion_postes*(origen-1)
y_movimiento = alto_postes+30
```

Para realizar la animación vamos a pasarle a la posición X e Y del disco un arreglo, que como se explicó anteriormente hace que el actor (disco) se mueva de forma continua pasando por los puntos indicados. Algo que no se mencionó con anterioridad es que, además de la lista de coordenadas, si se pasa otro número separado por coma a la posición X o Y, pilas lo interpreta a este número como el tiempo que debe demorar el movimiento hasta cada uno de los puntos. La realizar_movimiento(origen,destino) que tenemos hasta ahora es la siguiente:

```
tiempo_movimiento = 0.6;
def realizar_movimiento(origen,destino):
    if validar_movimiento(origen,destino):

        # realizar animacion
        x_inicial = separacion_postes*(origen-1)
        x_final = separacion_postes*(destino-1)
        y_movimiento = alto_postes+30
        y_final = alto*len(juego[destino])
        disco_a_mover = juego[origen][-1]
        discos[disco_a_mover].x = [x_inicial,x_final,x_final] , tiempo_movimiento/3
        discos[disco_a_mover].y = [y_movimiento,y_movimiento,y_final] ,
tiempo_movimiento/3

        # actualizar el estado del juego
```

```
    disco_a_mover = juego[origen].pop()
    juego[destino].append(disco_a_mover)

    pilas.avisar(str(juego))

else:
    pilas.avisar(u"Movimiento inválido")
print(juego)
```