PRÁCTICO Nº 1: MI PRIMER JUEGO

Objetivo:

Dar los primeros pasos con Pilas.

Temática:

El juego consiste en un pequeño torreta que ha de disparar a los monos que se generan al azar en pantalla y que intentan llegar hasta él para destruirlo. El juego incluye un sencillo marcador de puntuación, un control de sonido, un sistema de bonus (premios) y avisos de texto en pantalla.

El **primer paso** que vamos a dar es situar el escenario, es decir, elegir un fondo, preparar un marcador y poner un control de sonido. Para ello abrimos el editor de pila y escribimos el siguiente programa:

```
# Ejemplo en pilas engine: disparar_a_monos.py.
#
# Paso 1:   Vamos a situar el escenario, es decir,
#                elegir un fondo, preparar un marcador
#                y añadir un control de sonido.
#
# Copyright 2010 - Hugo Ruscitti
# License: LGPLv3 (see http://www.gnu.org/licenses/lgpl.html)
#
# Website - http://www.pilas-engine.com.ar

# UTF-8: especifica el estándar del formato de codificación de caracteres,
#           es necesario para que se muestren la ñ y los acentos
# -*- encoding: utf-8 -*-

import pilasengine

# Inicializar pilasengine, crea la ventana por defecto del juego 640x480 pixeles.
# A partir de aquí para acceder al motor pilas utilizamos solo el identificador "pilas"
pilas = pilasengine.iniciar()
```

```
# Usar el fondo "Pasto" que viene predefinido en pilas
pilas.fondos.Pasto()

# Añadir un marcador al juego, lo ubicamos en la esquina superior derecha,
# de color blanco. El marcador es un actor predefinido en pilas de tipo Puntaje.
# Los actores en pilas son objetos que aparecen en pantalla,
# tiene una posición determinada y se pueden manipular.
puntos = pilas.actores.Puntaje(x=230, y=200, color=pilas.colores.blanco)

# El objeto puntos lo utilizamos para cambiar las propiedades del marcador.
# Ampliar el tamaño del marcador.
puntos.magnitud = 40

# Añadir el conmutador de Sonido al juego,
# Esto permite activar/desactivar el sonido haciendo click sobre él.
pilas.actores.Sonido()

# Arrancar el juego: ejecutar se encarga de responder a las acciones del jugador,
# detectar las incidencias en el juego y mostrar imágenes y sonido.
pilas.ejecutar()
```

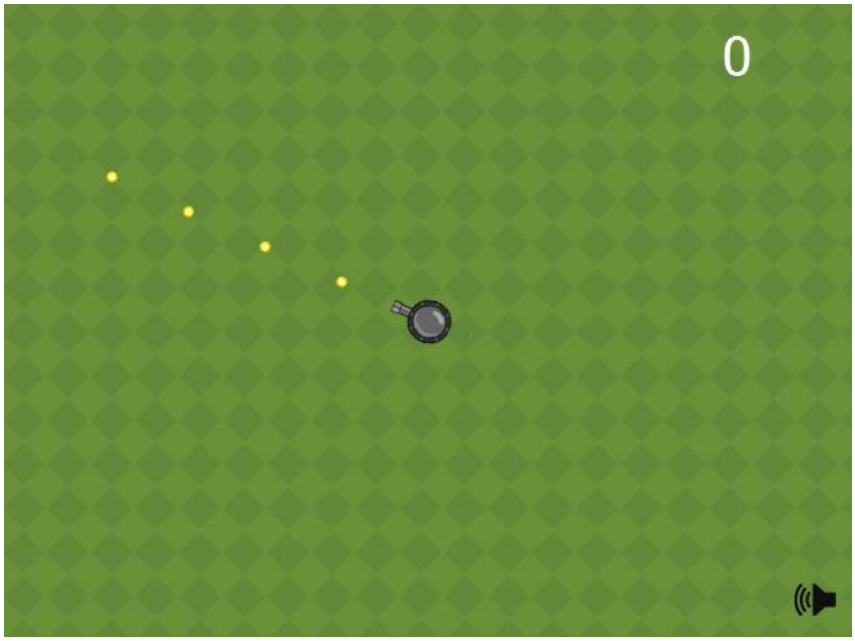
Luego guardamos el archivo con el nombre de **"paso1.py"**, al ejecutarlo debe aparecer la siguiente pantalla:

Pasemos al **pasó dos**, vamos a añadir ahora la torreta que va a manejar el jugador. Para facilitar la lectura e interpretación solo dejamos los comentarios de las líneas de código que se agregaron nuevas:

```
# Paso 2:   Vamos a añadir ahora la torreta que va
#           a manejar el jugador.
import pilasengine

pilas = pilasengine.iniciar()

pilas.fondos.Pasto()

puntos = pilas.actores.Puntaje(x=230, y=200, color=pilas.colores.blanco)
puntos.magnitud = 40

pilas.actores.Sonido()

# Para crear el objeto torreta debemos pasarles los siguientes argumentos:
```

```
# el tipo de munición que va a emplear, la lista de enemigos,
# la función que se debe llamar cuando colisionan una munición y un enemigo.

# Definimos un objeto del tipo actor Bala pero utilizamos pilasengine
# en vez de pilas, esto es porque solo queremos indicar el tipo de actor que
# vamos a utilizar pero no vamos a crearlo todavía porque eso no usamos Bala()
municion_bala_simple = pilasengine.actores.Bala

# Lista de enemigos vacía, es el grupo de enemigos a los que se va a disparar.
enemigos = []

# Función que se llamará cuando la munición que disparamos impacte
# con los enemigos. Por ahora no hace nada, usando pass permite dar
# paso a las demás instrucciones del programa.
def enemigo_destruido():
    pass

# Creamos el objeto torreta, obsérvese que al argumento cuando_elimina_enemigo
# le pasamos enemigo_destruido y no enemigo_destruido() ya que queremos pasarle
# la función que ha de usarse y no el resultado de ejecutarla.
torreta = pilas.actores.Torreta(municion_bala_simple=municion_bala_simple,
                                enemigos=enemigos,
                                cuando_elimina_enemigo=enemigo_destruido)

pilas.ejecutar()
```

Guarda el código con el nombre **"paso2.py"**, al ejecutarlo debe aparecer la siguiente pantalla:



El **tercer paso** va a ser añadir la aparición de los monos, nuestros enemigos en el juego, y su movimiento.

```
# Paso 3:   Añadir la aparición de nuestros enemigos
#           en el juego y su movimiento.
import pilasengine
import random

pilas = pilasengine.iniciar()

pilas.fondos.Pasto()

puntos = pilas.actores.Puntaje(x=230, y=200, color=pilas.colores.blanco)
```

```
puntos.magnitud = 40

pilas.actores.Sonido()

municion_bala_simple = pilasengine.actores.Bala

enemigos = []

def enemigo_destruido():
    pass

torreta = pilas.actores.Torreta(municion_bala_simple=municion_bala_simple,
                                enemigos=enemigos,
                                cuando_elimina_enemigo=enemigo_destruido)

# Velocidad de los enemigos se acercan a la torreta.
# Esta variable se usará para hacer más fácil o más difícil el juego.
tiempo = 6
# Variable booleana (bandera) que controlará si el juego ha terminado o no.
fin_de_juego = False

# Cada vez que se llame hay que crear un nuevo enemigo
def crear_enemigo():
    # Crear un objeto enemigo del tipo de actor mono
    enemigo = pilas.actores.Mono()

    # No queremos que el enemigo simplemente aparezca, sino que lo haga
    # con un efecto vistoso. Haremos que el enemigo aparezca gradualmente,
    # aumentando de tamaño. Para ello vamos a poner el atributo escala del enemigo
    # creado a 0, de esta manera conseguimos que inicialmente no se visualice.
    enemigo.escala = 0

    # La función pilas.utils.interpolar() es capaz de generar animaciones
    # muy vistosas y variadas. Sus argumentos son:
    # 1)Indica el actor que se va a interpolar
    # 2)Indica que atributo del actor va a modificarse, en este caso la escala.
    # 3)Valor final de la interpolación 0.5.
    # 4)Duración: es el tiempo que dura ese efecto, en nuestro caso, medio segundo.
    # 5)Tipo de animación, para este ejemplo elegimos 'elastico'.
    # Esta animación hace que el enemigo aumente la escala (tamaño) de 0 al 0.5,
    # es decir, a la mitad del tamaño original de la imagen del enemigo,
    # ya que el actor Mono tal como está predefinido en Pilas es muy grande
    # para el tamaño de la ventana actual.
    pilas.utils.interpolar(enemigo, 'escala', 0.5, duracion=0.5, tipo='elastico')

    # Dotar al enemigo con la habilidad "PuedeExplotar" al ser alcanzado
    # por un disparo. Esto se consigue a través del método aprender de los actores.
    enemigo.aprender("PuedeExplotar")

    # Finalmente, actualizamos la posición del mono modificando enemigo.x y enemigo.y
    # La función calcular_posicion() nos da una posición al azar
    enemigo.x,  enemigo.y  = calcular_posicion()

    # Añadirlo a la lista de enemigos.
    enemigos.append(enemigo)

    # Lista de los movimientos que utilizaremos en la función interpolar.
    tipo_interpolacion = ["lineal",
                          "aceleracion_gradual",
                          "desaceleracion_gradual",
                          "gradual"]

    # Con la función random.choice() para elegir uno de un tipo de movimiento
    # de la lista al azar
    interpolacion = random.choice(tipo_interpolacion)
```

```
    # Dotar al enemigo de un movimiento irregular más impredecible
    # En este caso vamos a interpolar el atributo 'x' e 'y' del actor.
    # De manera que el mono se acerque al centro donde se encuentra la torreta
    # con diferentes animaciones. En duración en  que un enemigo llegue a la torrera
    # esta definida en la variable tiempo=6 segundos.
    pilas.utils.interpolar(enemigo, 'x', 0, duracion=tiempo, tipo=interpolacion)
    pilas.utils.interpolar(enemigo, 'y', 0, duracion=tiempo, tipo=interpolacion)


    # Mientras dure el juego, se tienen que crear monos (hay que devolver True) y
    # cuando éste finalice, no (hay que devolver False).
    if not fin_de_juego:
        return True
    else:
        return False

# Función que genera las coordenadas x e y del enemigo para situarlo en una
# posición aleatoria en la ventana.
def calcular_posicion():
      # Para esto utilizamos la función randrange()
      # que devuelve un número al azar entre los dos dados.
    x = random.randrange(-320, 320)
    y = random.randrange(-240, 240)

    # Para evitar que el enemigo aparezca demasiado cerca de la torreta y
      # haga el juego imposible, si las coordenadas generadas son menores de 100,
      # se le aleja una distancia de 180.
    if x >= 0 and x <= 100:
        x = 180
    elif x <= 0 and x >= -100:
        x = -180

    if y >= 0 and y <= 100:
        y = 180
    elif y <= 0 and y >= -100:
            y = -180

    # devuelve la posición x e y donde se ubicará el actor
    return  x,y



# La manera de conseguir con Pilas que se realice una tarea cada cierto tiempo
# es usar la función tareas.siempre(1, crear_enemigo), debemos indicarle el
# tiempo en segundos que queremos que pase cada vez que se realice la tarea
# y la función que queremos que se invoque en este caso "crear_enemigo"
# cada un segundo. Para que la tarea siga ejecutandose deben devolver True.
# Un juego más elaborado consta de diferentes niveles, pantallas,
# presentaciones, en el siguiente practico explicaremos el método escena_actual().
pilas.escena_actual().tareas.siempre(1, crear_enemigo)

pilas.ejecutar()
```

Guarda el código con el nombre **"paso3.py"**, al ejecutarlo debe aparecer la siguiente pantalla donde los enemigos aparecen y se dirigen hacia el centro donde se encuentra la torreta:

Paso cuatro, monos atacando, torreta disparando... ¡Algo tiene que pasar cuando choquen! En este paso vamos a dedicarnos precisamente a eso, a implementar la destrucción de los monos y de la torreta cuando éstos lo alcancen.

```
# Paso 4:   implementar la destrucción de los monos
#           y de la torreta cuando éstos lo alcancen.

import pilasengine
import random

pilas = pilasengine.iniciar()

pilas.fondos.Pasto()

puntos = pilas.actores.Puntaje(x=230, y=200, color=pilas.colores.blanco)
puntos.magnitud = 40

pilas.actores.Sonido()

municion_bala_simple = pilasengine.actores.Bala

enemigos = []

# Función que se llamará cuando la munición que disparamos impacte
# con un enemigo. Sus parámetros son disparo y enemigo.
def enemigo_destruido(disparo, enemigo):
      # El mono al ser alcanzado por una munición explota pero no se elimina,
      # entonces se debe eliminar el mono con el método eliminar().
      enemigo.eliminar()
      # La munición también se debe eliminar al colisionar con un enemigo.
      disparo.eliminar()
      # Actualizar en uno la puntuación del marcador con el método aumentar(1),
      # pero lo haremos con efecto gradualmente, aumentando el tamaño de 0 a 1,
      # en medio segundo y con tipo de animación 'elastico'.
      puntos.escala = 0
      pilas.utils.interpolar(puntos, 'escala', 1, duracion=0.5, tipo='elastico')
      puntos.aumentar(1)
```

```
torreta = pilas.actores.Torreta(municion_bala_simple=municion_bala_simple,
                                enemigos=enemigos,
                                cuando_elimina_enemigo=enemigo_destruido)


tiempo = 6
fin_de_juego = False

def crear_enemigo():
    enemigo = pilas.actores.Mono()

    enemigo.escala = 0
    pilas.utils.interpolar(enemigo, 'escala', 0.5, duracion=0.5, tipo='elastico')

    enemigo.aprender("PuedeExplotar")

    enemigo.x,  enemigo.y  = calcular_posicion()

    enemigos.append(enemigo)

    tipo_interpolacion = ["lineal",
                          "aceleracion_gradual",
                          "desaceleracion_gradual",
                          "gradual"]

    interpolacion = random.choice(tipo_interpolacion)

    pilas.utils.interpolar(enemigo, 'x', 0, duracion=tiempo, tipo=interpolacion)
    pilas.utils.interpolar(enemigo, 'y', 0, duracion=tiempo, tipo=interpolacion)


    if not fin_de_juego:
        return True
    else:
        return False

def calcular_posicion():
    x = random.randrange(-320, 320)
    y = random.randrange(-240, 240)

    if x >= 0 and x <= 100:
        x = 180
    elif x <= 0 and x >= -100:
        x = -180

    if y >= 0 and y <= 100:
        y = 180
    elif y <= 0 and y >= -100:
            y = -180
    return  x,y



# El jugador pierde cuando no es capaz de eliminar a todos los monos y uno de ellos
# alcanza la torreta.
def perder(torreta, enemigo):
      # Definimos la variable como global para indicar que hacemos referencia a la
      # variable anteriormente definida en el programa.
      global fin_de_juego
      # El Mono muestra una sonrisa victoriosa en su cara
      enemigo.sonreir()
      # Invoca a la función eliminar() del actor torreta
      torreta.eliminar()
      # Eliminamos todas las tareas que se están realizando
```

```
        pilas.escena_actual().tareas.eliminar_todas()
        # Indicar fin de juego
        fin_de_juego = True
        # Mostramos un breve mensaje en la parte inferior de la ventana,
        # con el puntaje conseguido a través del método obtener() del actor Puntaje.
        pilas.avisar("GAME OVER. Conseguiste %d puntos" %(puntos.obtener()))

pilas.escena_actual().tareas.siempre(1, crear_enemigo)

# Cuando un mono colisiona con la torreta, el juego tiene que realizar las tareas
# que se encargan de darlo por terminado. Para ello, usamos método colisiones.agregar()
# predefinido en Pilas para añadir un tipo de colisión y su respuesta. Este método
# genera que se invoque la función que le pasamos como tercer argumento perder() cuando
# colisionen los actores indicados en los otros dos argumentos (torreta y enemigos).
pilas.escena_actual().colisiones.agregar(torreta, enemigos, perder)

pilas.ejecutar()
```

Guarda el código con el nombre **"paso4.py"**, al ejecutarlo debe aparecer la siguiente pantalla:

El **pasó 5**, nos quedan los detalles que hacen el juego más visual y algo más variado; mensajes, dificultad, bonus:

```
# Paso 5:   Se implementan detalles que hacen el juego más
#                 visual, mensajes, nivel de dificultad, bonus.
import pilasengine

import random

pilas = pilasengine.iniciar()

pilas.fondos.Pasto()

puntos = pilas.actores.Puntaje(x=230, y=200, color=pilas.colores.blanco)
puntos.magnitud = 40

pilas.actores.Sonido()

municion_bala_simple = pilasengine.actores.Bala
```

```
# Definimos un objeto del tipo actor BalasDoblesDesviadas para asignar como
# bonus cuando destruye una estrella
municion_doble_bala = pilasengine.actores.BalasDoblesDesviadas

enemigos = []

def enemigo_destruido(disparo, enemigo):
      enemigo.eliminar()
      disparo.eliminar()

      puntos.escala = 0
      pilas.utils.interpolar(puntos, 'escala', 1, duracion=0.5, tipo='elastico')
      puntos.aumentar(1)


torreta = pilas.actores.Torreta(municion_bala_simple=municion_bala_simple,
                                enemigos=enemigos,
                                cuando_elimina_enemigo=enemigo_destruido)


tiempo = 6
fin_de_juego = False

def crear_enemigo():
    enemigo = pilas.actores.Mono()

    enemigo.escala = 0

    pilas.utils.interpolar(enemigo, 'escala', 0.5, duracion=0.5, tipo='elastico')

    enemigo.aprender("PuedeExplotar")

    enemigo.x,  enemigo.y  = calcular_posicion()

    enemigos.append(enemigo)

    tipo_interpolacion = ["lineal",
                          "aceleracion_gradual",
                          "desaceleracion_gradual",
                          "gradual"]

    interpolacion = random.choice(tipo_interpolacion)

    pilas.utils.interpolar(enemigo, 'x', 0, duracion=tiempo, tipo=interpolacion)
    pilas.utils.interpolar(enemigo, 'y', 0, duracion=tiempo, tipo=interpolacion)

    if not fin_de_juego:
        return True
    else:
        return False

def calcular_posicion():
    x = random.randrange(-320, 320)
    y = random.randrange(-240, 240)

    if x >= 0 and x <= 100:
        x = 180
    elif x <= 0 and x >= -100:
        x = -180

    if y >= 0 and y <= 100:
        y = 180
    elif y <= 0 and y >= -100:
            y = -180
```

```
    return  x,y

def perder(torreta, enemigo):
      global fin_de_juego

      enemigo.sonreir()
      torreta.eliminar()
      pilas.escena_actual().tareas.eliminar_todas()
      fin_de_juego = True
      pilas.avisar("GAME OVER. Conseguiste %d puntos" %(puntos.obtener()))


# Asigna a la torreta munición simple.
def asignar_arma_simple():
    torreta.municion = municion_bala_simple
    # Esta función no devolveremos True para que, así, se ejecute una única vez.

# Asigna a la torreta munición doble.
def asignar_arma_doble(estrella, disparo):
    # Cambiar la munición de la torreta a balas_dobles
    torreta.municion = municion_doble_bala
    # Eliminar la estrella a la que hemos disparado y acertado
    estrella.eliminar()
    #darle una temporalidad a la munición extra que acabamos de activa
    # A los 10 segundos se ejecuta la función asignar_arma_simple()
    # que hace lo propio, devolviendo la torreta a su munición estándar.
    pilas.escena_actual().tareas.siempre(10, asignar_arma_simple)
    # Avisar de cambio de munición con un texto
    pilas.avisar("ARMA MEJORADA")
    # Esta función no devolveremos True para que, así, se ejecute una única vez.


# La función eliminar_estrella() (que solo hace eso mismo) no devuelve el valor
# True y que, por tanto, solo se ejecuta una vez
def eliminar_estrella(estrella):
    estrella.eliminar()

# Cada cierto tiempo aparece una estrella en pantalla que, si es destruida,
# cambia temporalmente la munición que usa la torreta a unas balas dobles.
def crear_estrella():
    # El argumento del if tiene la misión de generar un número aleatorio en el rango
    # del 0 al 10 y, solo si dicho número es mayor de 5, se procede a continuar.
    if random.randrange(0, 10) > 5:
        # No se debe crear una estrella si ya estamos en periodo de bonus,
        # ya que ya hemos sido premiados. Para ello, vamos a utilizar la función
        # de Python issubclass(). Esta función toma dos argumentos y devuelve True
        # si son instancias de la misma clase. Por lo tanto, devolverá True
        # (y en consecuencia, se continuará con la ejecución del contenido del
        # bloque if) si la torreta está usando la munición de balas simples.
        # La torreta posee la habilidad de DispararConClick una determinada
        # munición que es la que estamos chequeando.
        if issubclass(torreta.habilidades.DispararConClick.municion,
                      municion_bala_simple):
            x, y  = calcular_posicion()

            # crear la estrella, y lo hacemos en una posición al azar
            estrella = pilas.actores.Estrella(x,y)
            # Utilizamos la misma animación que el mono
            pilas.utils.interpolar(estrella, 'escala', 0.5, duracion=0.5,
                                   tipo='elastico')

            # Permitir que la torreta pueda destruirla, cuando la estrella colisione
            # con uno de los torreta.habilidades.DispararConClick.proyectiles se
            # lanzará la función asignar_arma_doble().
```

```
            pilas.escena_actual().colisiones.agregar(estrella,
                                    torreta.habilidades.DispararConClick.proyectiles,
                                    asignar_arma_doble)

            # Eliminarla pasado un tiempo, cada 3 segundos se lanzará la función
            # eliminar_estrella() que recibe como argumento el objeto estrella.
            pilas.escena_actual().tareas.siempre(3, eliminar_estrella, estrella)

def reducir_tiempo():
    #Indicamos que hacemos referencia a la variable definida en el programa.
    global tiempo

    # Disminuir la variable tiempo que acelera el movimiento de los monos
    tiempo -= 1
    # Avisamos al jugador que aumenta la dificultad a través de un aviso
    pilas.avisar("HURRY UP!!! Se vienen los monos.")
    # No queremos que la variable tiempo llegue a 0, entonces el mayor nivel
    # de dificultad es medio segundo.
    if tiempo < 1:
        tiempo = 0.5

    # Devolvemos True para asegurarnos que la tarea repetitiva no deje de realizarse
    return True

pilas.escena_actual().tareas.siempre(1, crear_enemigo)

# Crear un bonus (premio) para que el jugador cambie de munición.
# Invocar a la función crear_estrella cada 5 segundos.
pilas.escena_actual().tareas.siempre(5, crear_estrella)

# Aumentar la dificultad del juego cada 20 segundos.
# Agregamos la tarea de, cada 20 segundos, lanzar la función reducir_tiempo().
pilas.escena_actual().tareas.siempre(20, reducir_tiempo)

pilas.escena_actual().colisiones.agregar(torreta, enemigos, perder)

# Agregamos un aviso inicial con las instrucciones de juego, indicando que se
# usa el ratón y se dispara al hacer click. La "u" delante de la cadena de texto
# significa versión unicode de caracteres y evita que se muestren caracteres
# extraños al usar el acento en 'ratón'.
pilas.avisar(u"Mueve el ratón y haz click para destruirlos.")

pilas.ejecutar()
```