PRÁCTICO Nº 3: CREANDO MIS ACTORES

Objetivo:

Crear nuestros propios actores**.**

Temática:

Cuando tu juego crece, comienza la necesidad de crear tus propios actores, darles personalidad y lograr funcionalidad personalizada. En base a los actores prediseñados de pilas, vamos a crear nuestros propios actores para utilizar en el juego disparar monos.

Actores personalizados

Los actores son una pieza clave de pilas, nos sirven para representar a los personajes dentro de videojuego, con sus propios atributos, apariencia y comportamiento.

Un ejemplo, podemos crear un actor personalizado "**Enemigo**", que hereda de la clase Actor de pilas y asignarle una imagen diferente e incorporar las propiedades y métodos que habíamos definidos para los monos de nuestro juego.

```
from pilasengine.actores.actor import Actor
class Enemigo(pilasengine.actores.Actor):

      def iniciar(self, tiempo):
            self.imagen = "data/imagenes/piedra_grande.png"

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

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


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


            # 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 duracion en  que un enemigo
            # llegue a la torrera esta definida en la variable tiempo=6 segundos.
            self.pilas.utils.interpolar(self, 'x', 0, duracion=tiempo,
                                        tipo=interpolacion)
            self.pilas.utils.interpolar(self, 'y', 0, duracion=tiempo,
                                        tipo=interpolacion)

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

pilas.actores.vincular(Enemigo)
```

Se invoca a través de la siguiente sentencia:

```
# Crear un objeto enemigo del tipo de actor Enemigo
enemigo = self.pilas.actores.Enemigo(self.tiempo)
```

Otro ejemplo, en nuestro juego podríamos crear un actor personalizado "**Protagonista**", que hereda de la clase Torreta de pilas y asignarle una imagen diferente.

```
from pilasengine.actores.torreta import Torreta
class Protagonista(Torreta):

      def iniciar(self, municion_bala_simple=None, enemigos=[],
                  cuando_elimina_enemigo=None, x=0, y=0,
                  frecuencia_de_disparo=10):
            Torreta.iniciar(self,municion_bala_simple, enemigos,
                            cuando_elimina_enemigo,x,y,
                            frecuencia_de_disparo)
            self.imagen = "data/imagenes/pampa.png"
            self.aprender("PuedeExplotarConHumo")

pilas.actores.vincular(Protagonista)
```

Se invoca a través de la siguiente sentencia:

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

Otras características (Opcional)

Agregaremos otras características a los actores personalizados, es opcional.

a. Crear un actor personalizado para estrella, implementar el actor personalizado en el juego y asignarle una imagen diferente.
b. Crear un actor personalizado para municion_doble, implementar el actor personalizado en el juego y asignarle una imagen diferente.