PRÁCTICO Nº 2: ORGANIZANDO MÍ JUEGO

Objetivo:

Organizar el juego en torno a escenas y el concepto de modularización**.**

Temática:

En base al juego disparar monos del Practico N° 1 vamos a agregarle música y un menú de presentación con tres opciones "jugar", "ayuda" y "salir". Esto nos permitirá aprender nuevas técnicas, empezando por la modularización y la organización del juego en torno a escenas.

Modularización

La modularización intenta identificar partes reutilizables de un programa y desacoplarlas en archivos diferentes. Por ejemplo, si el comportamiento del mono lo implementamos por separado, más adelante podemos cambiar la forma en la que se crean, se desplazan o se destruyen sin interferir en nada más del videojuego. Además, permite que el código sea manejable, al estar divido en piezas más pequeñas y no un solo archivo de una longitud inabarcable.

¡Organizarse el algo muy importante! Cuanto más organizado este nuestro videojuego el código fuente será más fácil de compartir, modificar y mantener. Así que nosotros vamos a crear una carpeta **data** que contiene los archivos multimedia.

Escena

El concepto de escena es una manera más apropiada de dividir un videojuego en secciones diferentes sin que interfieran unas con otras y guardando el estado en el que se encuentran.

Consigna:

Buscaremos separar el código del menú en escenas. Las tres escenas que constará el juego son:

a. **Escena Menú** es la pantalla de inicio, con el menú de selección del juego que da acceso a las demás partes.

b. **Escena ayuda** que contiene la pantalla con las breves instrucciones para el uso del juego.

c. **Escena juego** es la implementación del juego propiamente dicho.

Para sacar provecho de las facilidades y recursos que ofrece pilas es necesario utilizar el paradigma de programación orientado a objetos. Un ejemplo de menú con tres escenas que utilice las capacidades de pilas podría ser el siguiente:

```
import pilasengine
import random

pilas = pilasengine.iniciar(ancho=800, alto=600, titulo="Disparar Monos")

# Todas las escenas derivan de una misma clase, la clase padre
# de todas las escenas: pilas.escena.Base

# Menu princial del juego
class EscenaMenu(pilasengine.escenas.Escena):
    # Es la escena de presentacion donde se elijen las opciones del juego.

    # En método iniciar() debemos colocar todo aquello que queremos que
    # aparezca al comenzar la escena
    def iniciar(self):
            # Definimos el fondo para la pantalla de ayuda.
            self.pilas.fondos.Noche()
            # Creamos las tuplas en las que el primer elemento
            # es el texto que se quiere mostrar y el segundo es
            # la función que se ejecutará al seleccionarlo.
            opciones = [
                  ("Comenzar a jugar", self.comenzar_a_jugar),
                  ("Ver ayuda", self.mostrar_ayuda_del_juego),
                  ("Salir", self.salir_del_juego)
                 ]
            # Utiliza el actor predefinido de Pilas pilas.actores.Menu
            # Indicamos la coordenada y donde queremos que aparezca.
            self.pilas.actores.Menu(opciones, y=-50)

    def comenzar_a_jugar(self):
            self.pilas.escenas.EscenaJuego()

    def salir_del_juego(self):
            self.pilas.terminar()

    def mostrar_ayuda_del_juego(self):
            self.pilas.escenas.EscenaAyuda()

# juego
class EscenaJuego(pilasengine.escenas.Escena):

    # 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

    def iniciar(self):
            self.crear_escenario()
            self.crear_personajes()
            self.jugar()

    def crear_escenario(self):
            self.pilas.fondos.Pasto()
            self.pilas.avisar("Pulsa ESC para regresar")
            self.pulsa_tecla_escape.conectar(self.cuando_pulsa_tecla_escape)
            # 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.
        self.puntos = self.pilas.actores.Puntaje(x=230, y=200,
                                                 color=self.pilas.colores.blanco)
        # El objeto puntos lo utilizamos para cambiar las propiedades del
        # marcador. Ampliar el tamaño del marcador.
        self.puntos.magnitud = 40

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

    def cuando_pulsa_tecla_escape(self, *k, **kv):
        # Si es la tecla ESC
        # Detener el tema
        self.sonido_de_musica.detener()
        # Volver a Menu
         self.pilas.escenas.EscenaMenu()


    def crear_personajes(self):
        # 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()
        self.municion_bala_simple = pilasengine.actores.Bala

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

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

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


    # Función que se llamará cuando la munición que disparamos impacte
    # con un enemigo. Sus parámetros son disparo y enemigo.
    def enemigo_destruido(self, 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'.
        self.puntos.escala = 0
        self.pilas.utils.interpolar(self.puntos, 'escala', 1, duracion=0.5,
```

```
                                        tipo='elastico')
            self.puntos.aumentar(1)

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

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


    def jugar(self):

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


MENSAJE_AYUDA = "Mueve el mouse y haz click para disparar."

# Ayuda
class EscenaAyuda(pilasengine.escenas.Escena):

    def iniciar(self):
                # Definimos el fondo para la pantalla de ayuda.
                self.pilas.fondos.Tarde()
                self.crear_texto_ayuda()
                # Habilitar el abandono de la pantalla de algún modo.
                self.pulsa_tecla_escape.conectar(self.cuando_pulsa_tecla_escape)

    # Añadir el Titulo y texto de la ayuda
    def crear_texto_ayuda(self):
                # Se escribe el texto "Instrucciones del juego"
                # en el centro a la altura y=200
                self.pilas.actores.Texto("Instrucciones del juego", y=200)
```

```
            # texto almacenado en la constante MENSAJE_AYUDA en otra posición
            self.pilas.actores.Texto(MENSAJE_AYUDA, y=0)
            # Muestra la advertencia de que se pulse la tecla escape para salir.
            self.pilas.avisar("Pulsa ESC para regresar")

    # La forma de declarar los argumentos de esta función es un
    # estándar en Python y son obligatorios
    def cuando_pulsa_tecla_escape(self, *k, **kv):
            #Volver a la escena principal, abandonando la ayuda.
            self.pilas.escenas.EscenaMenu()

# Vincular nuestras escenas a pilas
pilas.escenas.vincular(EscenaMenu)
pilas.escenas.vincular(EscenaJuego)
pilas.escenas.vincular(EscenaAyuda)

# Selecciona la escena Menu Principal
pilas.escenas.EscenaMenu()

pilas.ejecutar()
```

A menudo, el primer argumento de un método se llama **`self`** (uno mismo). Esto no es nada más que una convención de python Para poder acceder a las propiedades y métodos de una clase es necesario usar el argumento **`self`**.

Podemos observar que las propiedades tiempo y fin_de_juego de la clase menú están definidas dentro de la clase:

```
    tiempo = 6
    fin_de_juego = False
```

Esto permite acceder a las propiedades desde cualquier método de la clase, utilizando el argumento self.tiempo.

Dentro del método iniciar de la clase menú definimos los métodos que van a crear el escenario, definir los personajes y ejecutar el juego. Todos los métodos deben invocarse con el argumento self.

```
def iniciar(self):
            self.crear_escenario()
            self.crear_personajes()
            self.jugar()
```

Por otro lado, un ejemplo de las sentencias que debemos utilizar para implementar música en el juego:

```
# Carga el archivo musica.mp3 que se encuentra
# en la carpeta data y crea el objeto sonido_de_musica
sonido_de_musica = pilas.musica.cargar("data/musica.mp3")
# Repetir el tema indefinidamente
sonido_de_musica.reproducir(repetir=True)
# Detener el tema
sonido_de_musica.detener()
```

Hasta acá hemos implementado los pasos 1, 2 y 3 del juego, dejamos como tarea para el alumno la implementación del paso 4 y 5 del juego.

Otras funcionalidades (Opcional)

Agregaremos otras funcionalidades al juego, esta actividad es opcional.

a. Cuando aparezca una estrella que se escuche algún sonido, también cuando colisione con una munición que explote.

b. Provocar vibración de cámara cada vez se pierde una vida.

c. Vamos a agregarle al juego un **contador de vidas** representado por tres imágenes en la esquina superior izquierda, cada vez que un mono choca la torreta se debe eliminar una vida, tener en cuenta que al perder una vida se debe continuar en el mismo nivel de dificultad y con el mismo puntaje.

d. Ingresar un nombre de usuario en la pantalla inicial y luego mostrarlo al lado del puntaje en la pantalla siguiente.