Módulo Pygame

Este es un módulo pensado para crear videojuegos en Python que usamos para poder gestionar la interfaz y las entradas por teclado de forma más dinámica. Es un módulo bastante complejo así que este espacio lo usaremos para explicar las funciones que utilizamos nosotros.

Partamos de las dos clases más importantes, en nuestra opinión , la clase **Surface** (superficie) y la clase **Rect** (rectángulo).
En Pygame, la clase **Surface** y la clase **Rect** son dos componentes fundamentales para la creación de gráficos y la manipulación de imágenes en los juegos.

Clase Surface:
La clase **Surface** es la representación de una imagen o un área de píxeles en Pygame. Es una representación de una imagen o de un área de dibujo en memoria, donde puedes modificar los píxeles, cargar imágenes, dibujar formas y mucho más. Cada objeto **Surface** tiene un tamaño definido.
NOTA: Aunque a simple vista no parezca que usamos esta clase en el código muchas otras clases o funciones crean superficies aunque no se lea un **"pygame.surface"** en el código.

Métodos utilizados:
-blit(): La función **blit()** en Pygame es una de las más importantes cuando se trata de dibujar imágenes o superficies sobre otras superficies. Su propósito principal es copiar el contenido de una superficie fuente a una superficie destino en una ubicación específica. El término **"blit"** es una abreviatura de "**bit-block transfer**", que básicamente significa copiar un bloque de datos de una superficie a otra.
sintaxis:
```
surface.blit(source, dest, area = None, special_flags = 0)
```

- **surface:** Es la superficie de destino donde se copiará el contenido.
- **source:** Es la superficie fuente, es decir, la que contiene la imagen o el contenido que deseas copiar.
- **dest:** Las coordenadas **(x, y)** en la superficie de destino donde quieres que se coloque la parte de la superficie fuente. esta variable tiene que ser una tupla de la forma **(x, y)**.
- **area *(opcional)*:** Especifica un área rectangular de la superficie fuente que deseas copiar. Si no se especifica, se copia toda la superficie fuente.
- **special_flags *(opcional)***: Puedes aplicar ciertos efectos, pero normalmente se deja en cero.

Clase Rect:
La clase **Rect** en Pygame es un rectángulo y se utiliza para la colocación, detección de colisiones, o manipulación de superficies o áreas dentro de la ventana del juego. Es especialmente útil para trabajar con la posición y el tamaño de los elementos gráficos, como imágenes o sprites.

Atributos Importantes:
Un **Rect** tiene cuatro atributos principales: **x**, **y**, **width** y **height**.

La posición (x, y) representa la esquina o vértice superior izquierdo del rectángulo.
Por otro lado **width** y **height** representan las dimensiones o tamaño **en pixeles**.
Como crear un Rect:
Sintaxis: `rect = pygame.Rect(x, y, width, height)`

NOTA IMPORTANTE: Al crear un objeto de tipo **Rect** si o si se crea en base a su vértice superior izquierdo. luego se puede mover cambiando el resto de sus atributos pero al momento de crearlo se hace si o si desde ese vértice.

otros atributos:

- **top:** La coordenada vertical de la parte superior del rectángulo (**igual a y**).
- **bottom:** La coordenada vertical de la parte inferior del rectángulo (**y + height**).
- **left:** La coordenada horizontal de la parte izquierda del rectángulo (**igual a x**).
- **right:** La coordenada horizontal de la parte derecha del rectángulo (**x + width**).
- **topleft:** Una tupla que representa las coordenadas de la esquina superior izquierda (**x, y**).
- **topright:** Una tupla que representa las coordenadas de la esquina superior derecha (**x + width, y**).
- **bottomleft:** Una tupla que representa las coordenadas de la esquina inferior izquierda (**x, y + height**).
- **bottomright:** Una tupla que representa las coordenadas de la esquina inferior derecha (**x + width, y + height**).
- **center:** Una tupla que representa el centro del rectángulo (**x + width / 2, y + height / 2**).
- **size:** Una tupla que representa el tamaño del rectángulo (**width, height**).

NOTA: Relación entre **Surface** y **Rect**
La clase **Rect** es a menudo usada en conjunto con la clase **Surface** para posicionar superficies:
Cuando **blitteas (dibujas)** una superficie en la pantalla, se usa un **Rect** para especificar la posición y el área donde se dibujarán.
Por ejemplo, si tienes una superficie **surface** y un rectángulo **rect**, puedes usar **screen.blit(surface, rect)** para copiar la surface en la pantalla en la posición definida por el **rect**.

Métodos utilizados:
-get_width(): Retorna la anchura del Rectángulo
-get_height(): Retorna la altura del Rectángulo

Clase Image: En si imagen es un submódulo de **pygame** que proporciona funciones para cargar, manipular y guardar **imágenes**. La funcionalidad relacionada con imágenes se

centra principalmente en la gestión de superficies (**Surface**), que son la base para representar imágenes en Pygame.

Métodos utilizados:
-load(): Esta función carga una imagen desde un archivo y la convierte en una **superficie** (**Surface**) en Pygame, que luego puedes usar en el juego para dibujar en la pantalla, realizar transformaciones, etc. El formato puede ser PNG, JPG, BMP, entre otros.
Sintaxis: `imagen = pygame.image.load(ruta_imagen)`
NOTA: **ruta_imagen** debe contener la ruta al archivo que se quiere cargar.

Modulo draw:
En Pygame, el módulo **pygame.draw** proporciona una serie de funciones para dibujar formas geométricas sobre superficies, como rectángulos, círculos, líneas, etc. A pesar de que se utiliza sobre una superficie (por ejemplo, la pantalla o cualquier otra superficie que hayas creado), está como un módulo a parte ya que permite dibujar sobre muchos objetos que representan superficies, no solo sobre superficies del tipo **Surface**

Métodos utilizados:
rect(): La función **pygame.draw.rect()** se usa para dibujar un rectángulo (sólido o solo el contorno) en una superficie. El rectángulo puede ser de cualquier tamaño y puede colocarse en cualquier posición de la superficie.

sintaxis: `pygame.draw.rect(surface, color, rect, width = 0)`

- **surface**:
  - **Tipo:** Surface
  - **Descripción:** Es la superficie sobre la cual se dibujará el rectángulo. Esta puede ser la ventana principal de tu juego, o cualquier otra superficie sobre la que quieras dibujar el rectángulo.
- **color:**
  - **Tipo: tuple**
  - **Descripción:** Es el color con el que se dibujará el rectángulo. El color se define típicamente como una tupla de tres valores **RGB** (Rojo, Verde, Azul), cada uno en el rango de 0 a 255.
- **rect:**
  - **Tipo: Rect** o **tuple**
  - **Descripción:** Especifica la posición y tamaño del rectángulo. Este parámetro puede ser un objeto **Rect** de Pygame o una tupla con los 4 atributos principales de un objeto **Rect**: (x, y, width, height).
- **width(opcional):**
  - **Tipo: `int`**
  - **Descripción:** El grosor del borde del rectángulo. Si se establece en **0** (que es el valor predeterminado), el rectángulo será **sólido**(se rellenará por completo). Si se especifica un valor mayor a 0, se dibujará solo el **contorno** del rectángulo con ese grosor en píxeles.

Módulo transform:

En **Pygame**, el módulo **transform** se utiliza para realizar operaciones de transformación en **imágenes** o **superficies**. Esto incluye **redimensionar**, **rotar**, **reflejar** o realizar **distorsiones** en una **Surface** (que como vimos es el objeto principal donde se dibujan imágenes en Pygame)

NOTA IMPORTANTE:

- ***Relación con la clase Surface:*** *El módulo transform permite modificar una* ***Surface*** *de diferentes maneras. Entre ellas está el rotarla, escalarla, espejarla, etc.*
- ***Relación con el módulo image:*** *Recordemos que utilizamos image.load y este método devuelve una superficie para representar la imagen en pantalla*

Métodos utilizados:

-scale():

El método **pygame.transform.scale()** se utiliza para redimensionar una **superficie** (**Surface**) a un tamaño específico. El resultado es una nueva **Surface** con las dimensiones solicitadas.

NOTA: *No modifica la superficie original, sino que devuelve una nueva superficie con el tamaño modificado.*

sintaxis:
```
new_surface = pygame.transform.scale(surface, (ancho, alto))
```

- **surface:** Es la **superficie** (o imagen) que quieres redimensionar. Esta superficie puede ser una imagen cargada con **pygame.image.load()**, o cualquier otra **superficie** creada en el programa.
- **(ancho, alto):** Es una **tupla** que contiene las nuevas dimensiones de la superficie. Siendo el ancho y el alto a los que deseas cambiar la superficie.

-flip():

El método **pygame.transform.flip()** refleja una **superficie** (o imagen) a lo largo de los ejes **horizontal**(x) o **vertical**(y), creando una copia invertida de la superficie original.

sintaxis:
```
new_surface = pygame.transform.flip(surface, flip_x, flip_y)
```

- **surface:** La **superficie** (o imagen) que deseas **reflejar**(*"Efecto espejo"*). Este parámetro debe ser una **Surface** cargada o creada previamente en tu programa.
- **flip_x:** Un valor booleano (True o False) que determina si la imagen se reflejará horizontalmente (es decir, si se invertirá en el eje X).
- **flip_y:** Un valor booleano (True o False) que determina si la imagen se reflejará verticalmente (es decir, si se invertirá en el eje Y).

<u>Módulo display</u>:

El módulo **pygame.display** se encarga de gestionar la ventana de la aplicación, su configuración, y todo lo relacionado con la visualización en pantalla

Métodos utilizados:

-set_mode():

Este método es el más importante, ya que crea la **ventana** o pantalla donde se dibujará todo en **Pygame**. La **ventana** es de tipo **Surface**

sintaxis:
```
pygame.display.set_mode(size, flags = 0, depth = 0, display = 0, vsync = False)
```

- **size:** Una **tupla** (ancho, alto) que especifica el tamaño de la ventana.

- **flags**: Opciones adicionales para la ventana, como si debe ser de pantalla completa o no. Algunos valores comunes son:
    - **pygame.FULLSCREEN:** para pantalla completa.
    - **pygame.RESIZABLE:** para permitir redimensionar la ventana.
- **depth:** La profundidad de color de la pantalla. Normalmente, el valor predeterminado es adecuado.
- **display:** Si hay múltiples monitores, puedes especificar en cuál crear la ventana (por defecto es el primero).
- **vsync:** Si es True, habilita la sincronización vertical para evitar el "tearing" (desgarre de imagen: ayuda a la fluidez de los fotogramas para equipararla a la tasa de refresco del monitor).

-update():
Este método **actualiza** el contenido de la pantalla para reflejar cualquier cambio realizado en la **superficie**. Si solo deseas actualizar una **parte específica** de la pantalla, puedes pasar un **rectángulo (tupla (x, y, ancho, alto))**. Si no se pasa ningún argumento, se actualizará toda la pantalla.
sintaxis: `pygame.display.update(rect)`

- **rect(opcional):** Objeto **rect** que represente la zona que se quiere actualizar

<u>Módulo event</u>:
El módulo **pygame.event** es fundamental para gestionar y manejar eventos dentro de una aplicación o juego. Los eventos en Pygame son todas las interacciones que el usuario realiza con el sistema, como pulsaciones de teclas, clics de ratón, movimiento del ratón, entre otros. El módulo **event** permite detectar estos eventos y actuar en consecuencia.

Métodos utilizados:
-get(): el método **pygame.event.get()** devuelve una lista de todos los eventos que se han producido desde la última vez que se llamó. Los eventos se almacenan en forma de objetos de tipo **pygame.event.Event** y cada uno contiene información sobre el tipo de evento (tecla presionada, ratón movido, etc.) y otros detalles asociados a dicho evento.

<u>Módulo key</u>:
El módulo **pygame.key** maneja la entrada del teclado. Permite detectar qué teclas están siendo presionadas y cuál es su estado, lo que es fundamental para interactuar con el usuario en un juego o aplicación interactiva. Este módulo proporciona varias funciones y constantes que facilitan la detección de las teclas presionadas o liberadas.
Métodos utilizados:
-get_pressed(): Este método devuelve una lista de valores **booleanos** que indican el estado de todas las teclas. La lista tiene un elemento por cada tecla del teclado, donde **True** significa que la tecla está **presionada** y **False** que está **liberada**.

<u>Módulo time</u>:
El módulo **pygame.time** es una herramienta útil para controlar el **tiempo** y gestionar el ritmo de tu juego o aplicación interactiva. Te permite realizar tareas como la **sincronización de eventos**, la **medición de tiempo transcurrido**, la **espera de un tiempo determinado** o el **manejo de la velocidad de fotogramas** (frames por segundo, **FPS**).

Métodos utilizados:

-get_ticks(): El método **pygame.time.get_ticks()** devuelve el número de milisegundos transcurridos desde que se inició la ejecución de Pygame. Es una forma muy útil de medir el tiempo en tu aplicación o juego, ya que te permite calcular cuánto tiempo ha pasado desde que comenzó el programa, lo que puede ser utilizado para crear temporizadores, animaciones o efectos dependientes del tiempo.

Módulo mixer:
pygame.mixer es un submódulo que se encarga del manejo de sonido y música dentro de un videojuego. Sirve para cargar, reproducir, pausar y detener sonidos o música de fondo. Antes de usarlo, se debe inicializar con **pygame.mixer.init()** (aunque normalmente se inicializa con **pygame.init()** también).

Métodos utilizados:
<u>Para música</u>:
-load: carga un archivo de música. La ruta debe conducir a un archivo mp3.

```
Sonido = pygame.mixer.music.load(ruta_música)
```

-play(veces): Reproduce música las veces que se quiera. Si se coloca el parámetro en **-1** se reproduce infinitamente.

```
pygame.mixer.music.play(-1)
```

-stop: Detiene la música.

```
pygame.mixer.music.stop(
```

-set_volume(volumen): cambia el volumen en base al parámetro. Este va de 0.0 a 1.0 que equivale a silencio y el máximo del archivo

```
sonido.set_volume(0.5)
```

Cabe aclarar qie sonido ya tiene un archivo de audio cargado.

<u>Para efectos de sonido</u>:
-Sound: carga un efecto de sonido (debe ser .wav).

```
pygame.mixer.Sound(ruta_sonido)
```

-play(): reproduce el efecto

```
sonido.play()
```

-set_volume(volumen): cambia el volumen en base al parámetro. Este va de 0.0 a 1.0 que equivale a silencio y el máximo del archivo.

```
sonido.set_volume(0.5)
```