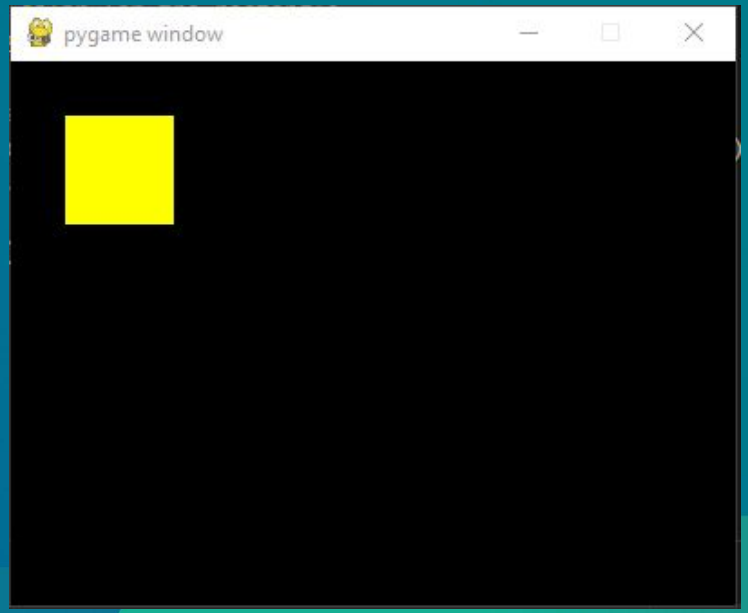
Facundo Falcone - Guevara Mariano - Manuel Samaniego

Biblioteca Pygame

Superficies y Rectángulos

Al utilizar **Pygame,** generalmente se utilizan superficies para representar la apariencia del objeto y su posición en la pantalla. Todos los objetos, textos e imágenes que creamos en Pygame se crean utilizando superficies.

Crear superficies en pygame es bastante fácil. Solo tenemos que pasar la altura y el ancho con una tupla al método **pygame.surface().**



Superficies y Rectángulos

Podemos utilizar varios métodos para formatear nuestra superficie como queramos. Por ejemplo, podemos utilizar **pygame.draw()** para dibujar formas, podemos utilizar el método **surface.fill()** para rellenar la superficie. Ahora, la implementación de estas funciones. Analicemos la sintaxis y los parámetros.

```
# Importing the library
import pygame
import time

# Initializing Pygame
pygame.init()

# Creating the surface
sample_surface = pygame.display.set_mode((400,300))

# Choosing red color for the rectangle
color = (255,255,0)

# Drawing Rectangle
pygame.draw.rect(sample_surface, color,
                 pygame.Rect(30, 30, 60, 60))

# The pygame.display.flip() method is used
# to update content on the display screen
pygame.display.flip()
```

Superficies y Rectángulos

- **surface (Surface)** -- superficie sobre la que dibujar.
- **color** (***Color*** *o int o tupla(int, int, int, [int])*) -- color para dibujar, el valor alfa es opcional si se usa una tupla (RGB[A]).
- **rect** (***Rect***) -- Rectángulo para dibujar, posición y dimensiones.
- **width** (*int*) – (opcional) se utiliza para el grosor de la línea o para indicar que se debe rellenar el rectángulo (no debe confundirse con el valor de ancho del parámetro rect).
  - if `width == 0`, (default) fill the rectangle
  - if `width > 0`, used for line thickness
  - if `width < 0`, nothing will be drawn

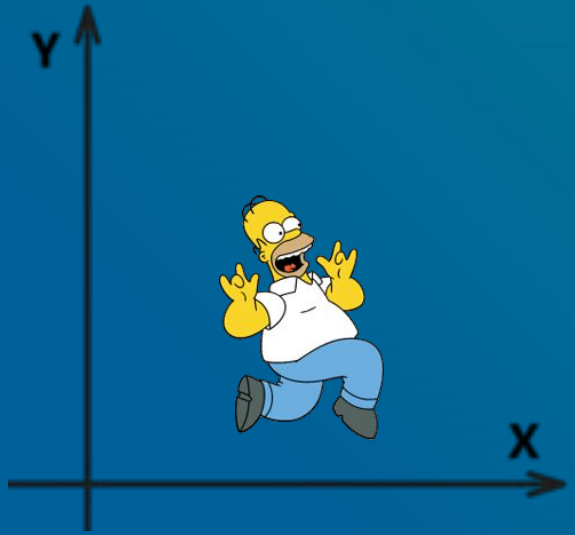
Superficies y Rectángulos

Veamos cómo dibujar un rectángulo con pygame.draw.rect.

```
rect(surface, color, rect) # Rect
rect(surface, color, rect, width=0, border_radius=0, border_top_left_radius=-1,
    border_top_right_radius=-1, border_bottom_left_radius=-1, border_bottom_right_radius=-1) # Rect
```

```
# Importing the library
import pygame
import time

# Initializing Pygame
pygame.init()

# Creating the surface
sample_surface = pygame.display.set_mode((400,300))

# Choosing red color for the rectangle
color = (255,255,0)

# Drawing Rectangle
pygame.draw.rect(sample_surface, color,
                 pygame.Rect(50, 30, 60, 60))

# The pygame.display.flip() method is used
# to update content on the display screen
while True:
    pygame.display.flip()
```

Movimiento



- El movimiento de nuestros objetos en pygame se logra actualizando las coordenadas de los mismos en los ejes X e Y dentro del bucle del juego
- Aumentar X mueve el objeto a la derecha, disminuirlo, hacia la izquierda
- Aumentar Y mueve el objeto hacia abajo, disminuirlo, hacia arriba

Movimiento

```
imagen_vertical = pygame.Surface((100,100)) #creo una superficie
imagen_vertical.fill(VERDE)
rectangulo_vertical = imagen_vertical.get_rect() # -> obtiene el rectangulo de la imagen

while True:
    clock.tick(FPS)
    for evento in pygame.event.get():
        if evento.type == pygame.QUIT:
            pygame.quit()
            sys.exit()

    PANTALLA.fill(NEGRO)
    PANTALLA.blit(imagen_vertical) # actualizo superficie en pantalla

    rectangulo_vertical.y += 10 # Aumento su posicion en el eje Y; avanza hacia abajo
    if rectangulo_vertical.top > ALTO:
        rectangulo_vertical.bottom = 0 # cuando llega al limite de la pantalla, lo mando arriba de nuevo

    pygame.display.flip()
```

Sonidos | Música de fondo

Para poder usar música en nuestro juego, necesitaremos importar el módulo **mixer** de la biblioteca Pygame
Una vez importado tenemos que inicializarlo llamando al método .init() para poder utilizarlo

```
import pygame.mixer as mixer
mixer.init()
```

Sonidos | Música de fondo

Una vez inicializado, en caso de querer cargar una música de fondo para nuestro juego, podemos usar en conjunto 3 métodos del módulo music perteneciente a mixer. Como argumento le pondremos la ruta al archivo de audio para cargarlo.

```
mixer.music.load('./assets/snd/select.mp3')
```

Sonidos | Música de fondo

Como siguiente paso configuraremos su volumen con valores entre 0 y 1, valiéndonos números flotantes, por ejemplo volumen al 40%

```
mixer.music.set_volume(0.4)
```

Este método nos permitirá modificar el volumen según lo necesitemos en nuestro juego, ya sea con algún control de volumen o simplemente aplicar un volumen por defecto que no varíe.

Sonidos | Música de fondo

Finalmente para reproducir el sonido usaremos el método play() invocandolo donde lo necesitemos

```
mixer.music.play()
```

Esto está diseñado para reproducir **archivos de música más largos** como canciones completas. Nos ofrece funciones específicas para controlar la reproducción de la música, tal como los métodos:
mixer.music .pause() | **mixer.music.unpause()** | **mixer.music.stop()**

La limitación de esto es que puede manejar un archivo a la vez, si se intenta cargar un nuevo archivo, el anterior se detendrá.

Sonidos | Efectos de sonido

Otra forma de hacerlo, por ejemplo si queremos poner sonidos a los personajes de nuestro juego, es mediante la clase Sound, cuyo primer argumento será la ruta a nuestro audio.

```
sonido = mixer.Sound(ruta_al_audio)
sonido.set_volume(0.4)
sonido.play()
```

Esto es ideal para efectos de sonido cortos, efectos de juego (disparos, explosiones, etc). Se pueden crear múltiples instancias de Sound y reproducir varios sonidos simultáneamente como también controlar el volumen de todas también de manera simultánea. Es una forma más flexible para controlar múltiples sonidos al mismo tiempo.