



# Programación y Laboratorio I

# pygame

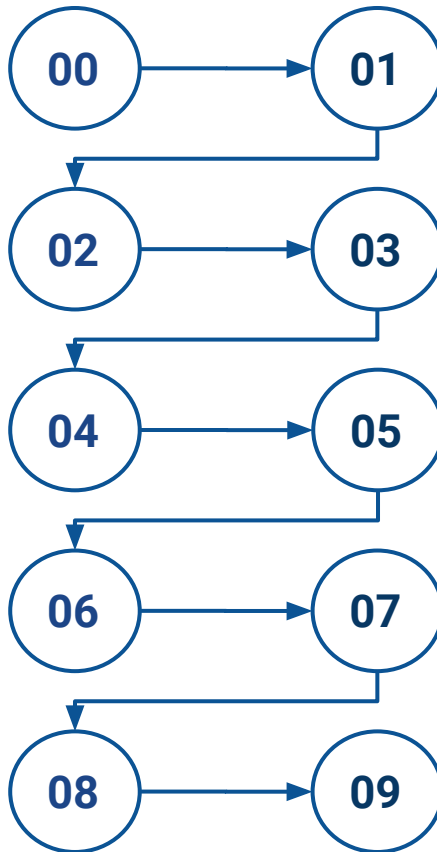
¿Qué es pygame?

Ventanas

Eventos

Textos

Colisiones



Características

Superficies

Imagenes

Rectangulos

Sonidos

# ¿Qué es pygame?

**Pygame** es un contenedor de Python para la biblioteca **SDL** , que significa Simple DirectMedia Layer .

SDL brinda acceso multiplataforma a los componentes de hardware multimedia subyacentes de su sistema, como sonido, video, mouse, teclado y joystick.

# ¿Qué es pygame?

```
import pygame

pygame.init() #Se inicializa pygame

screen = pygame.display.set_mode([500, 500]) #Se crea una ventana

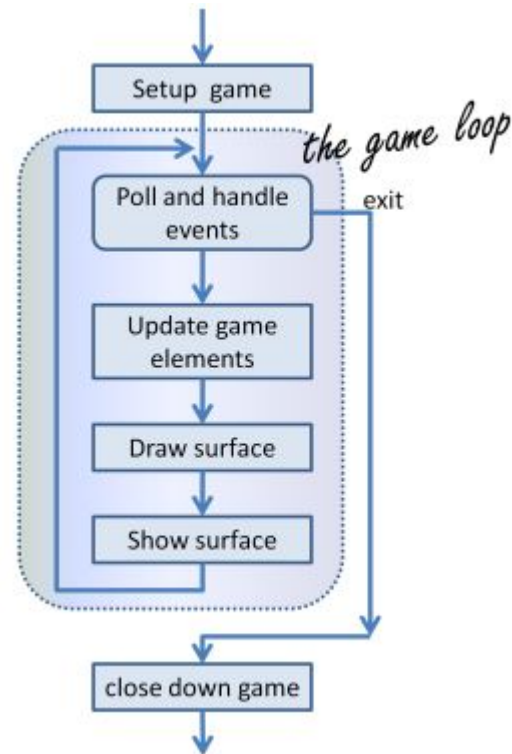
running = True

while running:

    # Se verifica si el usuario cerro la ventana
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    screen.fill((255, 255, 255))# Se pinta el fondo de la ventana
    # Se dibuja un círculo azul en el centro
    pygame.draw.circle(screen, (0, 0, 255), (250, 250), 75)
    pygame.display.flip()# Muestra los cambios en la pantalla

pygame.quit() # Fin
```



# Características

- Imágenes en formato PNG, BMP, ..
- Sistemas de sonido, formato WAV, OGC, MP3.
- Operaciones relacionadas con el gestor de ventanas.
- Eventos de aplicación
- Manejo de dispositivos de entrada
- Temporizadores.
- Colisiones, sistema de Sprites (objetos de un juego).

El módulo display permite controlar todo lo que tiene que ver con las ventanas y las pantallas de un programa.

```
#Se crea una ventana
```

```
screen = pygame.display.set_mode([500, 500])
```

```
# Título de la ventana
```

```
pygame.display.set_caption("Hola Mundo")
```

# Superficies

Un objeto **Surface** representa un buffer de memoria de píxeles, la pantalla se representa también como una superficie.

# Se pinta el fondo de la ventana de blanco

```
screen.fill((255, 255, 255))
```

# Superficies

Sobre una superficie se pueden dibujar diferentes formas (rectángulos, círculos, elipses, líneas, etc..)

# Se dibuja un círculo azul

```
pygame.draw.circle(screen, (0, 0, 255), (250, 250), 75)
```

# Se dibuja un cuadrado amarillo

```
pygame.draw.rect(screen, (255, 255, 0), (30, 30, 60, 60))
```



Pygame provee control sobre los dispositivos de entrada más comunes :

- Teclado (`pygame.key` )
- Mouse (`pygame.mouse`)
- Joystick (`pygame.joystick`)

Pygame maneja todos sus mensajes de eventos a través de una cola de eventos.

```
pygame.event.get()
```

La cola de eventos contiene objetos de evento  
`pygame.event.EventType`

```
# Se verifica si el usuario cerro la ventana  
for event in pygame.event.get():  
    print(type(event)) # <class 'Event'>
```

Todas instancias de Event contienen un identificador de tipo de evento y atributos específicos para ese tipo de evento.

```
# Se verifica si el usuario cerro la ventana
for event in pygame.event.get():
    if event.type == pygame.QUIT:
        running = False
```

# Eventos (mouse)

Se verifica si el evento es el del mouse presionado y luego se verifica cual es la posición:

```
for event in pygame.event.get():  
    if event.type == pygame.QUIT:  
        running = False  
  
    if event.type == pygame.MOUSEBUTTONDOWN :  
        print(event.pos) # (322, 153)
```

# Eventos (tiempo)

Para poder fijar un evento que ocurran por tiempo.

```
tick = pygame.USEREVENT + 0
pygame.time.set_timer(tick, 1000)
while running:
    for event in pygame.event.get():
        if event.type == tick:
            print("Ya paso un segundo")
```

# Eventos (teclado)

Se verifica si el evento es el de una tecla presionada y luego se verifica cual es la tecla:

```
for event in pygame.event.get():  
    if event.type == pygame.QUIT:  
        running = False  
  
    if event.type == pygame.KEYDOWN:  
        if event.key == pygame.K_x:  
            print("Se presiono la tecla X")
```

# Teclado (s/evento)

Se verifica si el evento es el de una tecla presionada y luego se verifica cual es la tecla:

```
from pygame.locals import K_x  
  
pressed_keys = pygame.key.get_pressed()  
  
if True in pressed_keys:  
    if pressed_keys[K_x]:  
        print("Se presiono la tecla X")
```



Con el método load de la clase image se puede generar una superficie desde una imagen

```
imagen = pygame.image.load("00.png")  
print(type(imagen)) # <class 'pygame.Surface'>
```

# Imágenes

Para copiar la superficie de la imagen en la superficie de la pantalla o en otra superficie se debe utilizar **blit()** indicando la ubicación

```
imagen = pygame.image.load("00.png")  
print(type(imagen)) # <class 'pygame.Surface'>  
screen.blit(imagen, (50, 50))
```

El método `render` me permite obtener una superficie a partir de un texto.

```
font = pygame.font.SysFont("Arial Narrow", 50)
text = font.render("HOLA MUNDO", True, (255, 0, 0))
print(type(text)) # <class 'pygame.Surface'>
```

Para copiar la superficie del texto en la superficie de la pantalla o en otra superficie se debe utilizar **blit()** indicando la ubicación

```
font = pygame.font.SysFont("Arial Narrow", 50)
text = font.render("HOLA MUNDO", True, (255, 0, 0))
screen.blit(text, (50, 50))
```

# Rectangulos

Pygame usa objetos **Rect** para almacenar y manipular áreas rectangulares.

Se crean a partir de las dimensiones de ancho, alto y la posición respecto al lado izquierdo y superior de un objeto.

`pygame.Rect (left, top, width, height)`

# Rectangulos

En cada juego, cada objeto requiere un conjunto de límites fijos que definen el espacio que ocupa.

Estos límites fijos son esenciales cuando el objeto interactúa o “choca” con otros objetos.

# Rectángulos

En Pygame se crean "**Rectángulos**" alrededor de los objetos para definir sus límites.



# Rectangulos

La forma más básica de crear un objeto Rect en Pygame es simplemente pasar dos tuplas.

La primera tupla (izquierda, arriba) representa la posición del Rect en la ventana, mientras que la segunda tupla (ancho, alto) representa las dimensiones del Rect.

```
# Se dibuja un cuadrado amarillo
```

```
rectangulo = pygame.Rect((30, 30), (60, 60))
```

```
pygame.draw.rect(screen, (255, 255, 0), rectangulo)
```



# Rectangulos

También es posible obtener un objeto Rect de una superficie y manipularlo.

```
imagen_dona = pygame.image.load("00.png")  
rect_dona = imagen_dona.get_rect()  
rect_dona.centerx = 200  
rect_dona.centery = 100  
print(rect_dona.width, rect_dona.height) # 200 200  
screen.blit(imagen_dona, rect_dona)
```

# Colisiones

La forma más sencilla de comprobar colisiones es ver si se solapan los rectángulos de las distintas superficies.



En Pygame se crean "**Rectángulos**" alrededor de los objetos para definir sus límites.

Para cada rectángulo, es posible comprobar si colisiona con otro.

```
if rect_player.colliderect(rect_piedra):  
    print("El jugador colisiona con la piedra")  
  
if rect_player.colliderect(rect_fuego):  
    print("El jugador colisiona con el fuego")
```

Es posible comprobar si un rectángulo colisiona con algún otro rectángulo de una lista de Rect.

```
if rect_player.collidelist(lista_rect) >= 0:  
    print("El jugador colisionó con la piedra")
```

También es posible comprobar si colisiona una superficie con una coordenada.

```
if event.type == pygame.MOUSEBUTTONDOWN :  
    if rect_player.collidepoint(event.pos) :  
        print("CLICK sobre el jugador")
```

# Sonidos

Los formatos de archivos de sonido que Pygame soporta son MID, WAV y MP3. Si al play se le pasa -1 como argumento, se va a reproducir el sonido de manera indefinida

```
pygame.mixer.init()
pygame.mixer.music.set_volume(0.7)
sonido_fondo = pygame.mixer.Sound("fondo.wav")
sonido_fondo.set_volume(0.2)
sonido_fondo.play()
sonido_fondo.stop()
```

# Ejemplos de TP Juego

- Ricardo Bartoncello

[GitHub](#)

[Video](#)

- Luciano Bardallo

[GitHub](#)

[Video](#)

- Santiago Collazo

[GitHub](#)

[Video](#)



# Programación y Laboratorio I