Documentación código:

Este script de Python utiliza la biblioteca Tkinter para crear una interfaz gráfica que muestra una lista de archivos de juegos (ROMs) y permite realizar acciones como ejecutar el juego seleccionado, borrar archivos, y copiar nuevos archivos desde una unidad USB:

1. Importación de módulos y bibliotecas necesarios:

Importa diversos módulos y bibliotecas necesarios para la interfaz gráfica y las funciones relacionadas, como os, tkinter, PIL (Python Imaging Library), subprocess, threading, time, pyudev, shutil, y pygame.

2. Función borrar_archivo_seleccionado:

```
def borrar_archivo_seleccionado():
    # Obtiene la selección actual en la lista
    seleccion = listbox_archivos.curselection()
    if seleccion:
        index = seleccion[0]
        nombre_archivo = listbox_archivos.get(index)
        ruta_completa = os.path.join(os.path.dirname(__file__), "roms", nombre_archivo)
        ruta_imagen = os.path.join(os.path.dirname(__file__), "roms",
f"{os.path.splitext(nombre_archivo)[0]}.png")
        try:
            # Intenta borrar el archivo y su imagen asociada
            os.remove(ruta_completa)
            os.remove(ruta_imagen)
            time.sleep(1)
            # Actualiza la lista y la imagen después de borrar
            listbox_archivos.delete(index)
            seleccionar_fila(None)
        except Exception as e:
            print(f"Error al borrar archivos: {e}")
```

1. **Obtiene la selección actual en la lista:**
    - Utiliza **curselection()** para obtener el índice de la selección actual en la lista de ROMs.
2. **Prepara las rutas de archivos:**
    - Construye las rutas completas del archivo y su imagen asociada utilizando el nombre del archivo.
3. **Intenta borrar el archivo y su imagen:**
    - Utiliza **os.remove()** para borrar el archivo y su imagen asociada.

4. **Espera un segundo después de borrar:**
    - Utiliza **time.sleep(1)** para esperar un segundo antes de realizar más acciones.
5. **Actualiza la lista y la imagen después de borrar:**
    - Utiliza **listbox_archivos.delete()** para eliminar el elemento seleccionado de la lista.
    - Llama a **seleccionar_fila(None)** para actualizar la imagen después de borrar.
6. **Maneja excepciones:**
    - Utiliza un bloque **try...except** para manejar cualquier excepción que pueda ocurrir durante el proceso de borrado.

3. Función cargar_nombres_archivos:

Retorna una lista de nombres de archivos en la carpeta de ROMs con extensiones .smc y .zip.

```
def cargar_nombres_archivos():
    carpeta_roms = os.path.join(os.path.dirname(__file__), "roms")
    if os.path.exists(carpeta_roms) and os.path.isdir(carpeta_roms):
        # Filtra los archivos con extensiones .smc y .zip
        nombres_archivos = [archivo for archivo in os.listdir(carpeta_roms) if archivo.lower().endswith((".smc", ".zip"))]
        return nombres_archivos
    else:
        return []
```

1. **Obtiene la ruta de la carpeta de ROMs:**
    - Construye la ruta completa de la carpeta de ROMs utilizando **os.path.join()**.
2. **Verifica la existencia y tipo de la carpeta:**
    - Utiliza **os.path.exists()** y **os.path.isdir()** para verificar si la carpeta de ROMs existe y es un directorio.
3. **Filtra los archivos con extensiones .smc y .zip:**
    - Utiliza una list comprehension para obtener los nombres de archivos que terminan con las extensiones deseadas.
4. **Retorna la lista de nombres de archivos:**
    - Retorna la lista resultante de nombres de archivos.

4. Función obtener_ruta_imagen_extension:

Retorna la ruta de la imagen asociada a la extensión del archivo.

```
def obtener_ruta_imagen_extension(extension):
```

```
if extension == ".smc":
    return os.path.join(os.path.dirname(__file__), "1.png")
elif extension == ".zip":
    return os.path.join(os.path.dirname(__file__), "2.png")
else:
    return os.path.join(os.path.dirname(__file__), "nofound.png")
```

1. **Determina la ruta de la imagen según la extensión del archivo:**
    - Utiliza una estructura **if...elif...else** para determinar la ruta de la imagen basándose en la extensión del archivo.
2. **Retorna la ruta de la imagen:**
    - Retorna la ruta completa de la imagen asociada a la extensión dada.

5. Función mostrar_imagen_centro:

Muestra la imagen en el centro de la interfaz.

```
def mostrar_imagen_centro(extension):
    ruta_imagen = obtener_ruta_imagen_extension(extension)
    imagen = Image.open(ruta_imagen)
    imagen_tk = ImageTk.PhotoImage(imagen)
    etiqueta_imagen_sistema.config(image=imagen_tk)
    etiqueta_imagen_sistema.image = imagen_tk
```

1. **Obtiene la ruta de la imagen:**
    - Utiliza la función **obtener_ruta_imagen_extension()** para obtener la ruta de la imagen según la extensión dada.
2. **Abre la imagen y la convierte a formato Tkinter:**
    - Utiliza **Image.open()** para abrir la imagen y **ImageTk.PhotoImage()** para convertirla a un formato compatible con Tkinter.
3. **Configura la etiqueta de la imagen del sistema:**
    - Utiliza **etiqueta_imagen_sistema.config()** para configurar la etiqueta de la imagen del sistema con la nueva imagen.

6. Función actualizar_imagen:

Actualiza la imagen según el archivo seleccionado.

```
def actualizar_imagen(nombre_archivo):
    ruta_imagen = os.path.join(os.path.dirname(__file__), "roms", f"{os.path.splitext(nombre_archivo)[0]}.png")
    if not os.path.exists(ruta_imagen):
        ruta_imagen = os.path.join(os.path.dirname(__file__), "nofound.png")
    imagen = Image.open(ruta_imagen)
    imagen_tk = ImageTk.PhotoImage(imagen)
    etiqueta_imagen.config(image=imagen_tk)
    etiqueta_imagen.image = imagen_tk
```

1. **Obtiene la ruta de la imagen según el nombre del archivo:**

- Construye la ruta de la imagen basándose en el nombre del archivo y la extensión ".png".

2. **Verifica la existencia de la imagen:**
   - Utiliza **os.path.exists()** para verificar si la imagen existe en la ruta especificada.
3. **Actualiza la ruta de la imagen si no existe:**
   - Si la imagen no existe, actualiza la ruta a la imagen de "nofound.png".
4. **Abre la imagen y la convierte a formato Tkinter:**
   - Utiliza **Image.open()** para abrir la imagen y **ImageTk.PhotoImage()** para convertirla a formato compatible con Tkinter.
5. **Configura la etiqueta de la imagen:**
   - Utiliza **etiqueta_imagen.config()** para configurar la etiqueta de la imagen con la nueva imagen.

7. Función seleccionar_fila:

Llamada al seleccionar una fila en la lista de ROMs.

Actualiza la imagen y muestra la imagen en el centro.

```
def seleccionar_fila(event):
    seleccion = listbox_archivos.curselection()
    if seleccion:
        index = seleccion[0]
        nombre_archivo = listbox_archivos.get(index)
        actualizar_imagen(nombre_archivo)
        _, extension = os.path.splitext(nombre_archivo)
        mostrar_imagen_centro(extension)
```

1. **Obtiene la selección actual en la lista:**
   - Utiliza **listbox_archivos.curselection()** para obtener el índice de la selección actual en la lista de ROMs.
2. **Verifica si hay una selección:**
   - Utiliza un bloque **if** para verificar si hay una selección.
3. **Obtiene el nombre del archivo seleccionado:**
   - Utiliza **listbox_archivos.get()** para obtener el nombre del archivo seleccionado.
4. **Actualiza la imagen según el archivo seleccionado:**
   - Llama a la función **actualizar_imagen()** con el nombre del archivo seleccionado como argumento.
5. **Obtiene la extensión del archivo:**
   - Utiliza **os.path.splitext()** para obtener la extensión del archivo.
6. **Muestra la imagen en el centro basándose en la extensión:**
   - Llama a la función **mostrar_imagen_centro()** con la extensión como argumento.

8. Función joystick_input:

Maneja la entrada del joystick para navegación y ejecución de comandos.

```
def joystick_input():
    verifica = 0
    conta = 0
    pygame.init()
    pygame.joystick.init()
    # Verificar si hay al menos un controlador conectado
    if pygame.joystick.get_count() == 0:
        print("No se encontraron controladores de Xbox conectados.")
        return
    joystick = pygame.joystick.Joystick(0)
    joystick.init()
    while True:
        if conta > 0:
            conta = conta - 1
        for event in pygame.event.get():
            if event.type == pygame.JOYAXISMOTION:

                # Mover hacia arriba o hacia abajo al mover el joystick izquierdo
                if event.axis == 1:  # Eje Y del joystick izquierdo
                    if event.value < -0.5 and conta == 0 and verifica == 0:
                        conta = 10000
                        mover_up()
                    elif event.value > 0.5 and conta == 0 and verifica == 0:
                        conta = 10000
                        mover_down()
            if event.type == pygame.JOYBUTTONDOWN:
                print(":{event.button}")
                # Presionar el botón A para ejecutar el comando
                if event.button == 0 and verifica == 0:  # Botón A
                    ejecutar_comando(None)
                    verifica = 1
                if event.button == 3 and verifica == 0:  # Botón y
                    borrar_archivo_seleccionado()
                    seleccionar_primer_archivo(None)
                if event.button == 4 and verifica == 1:  # Botón lb
                    verifica = 0
```

1. **Inicializa variables y módulos de Pygame:**
    - Inicializa las variables **verifica** y **conta**.
    - Inicializa Pygame y el módulo de joystick.
2. **Verifica la existencia de controladores de Xbox:**

- Utiliza **pygame.joystick.get_count()** para verificar si hay al menos un controlador conectado.

3. **Inicializa el controlador de joystick:**
   - Utiliza **pygame.joystick.Joystick(0)** para obtener el primer controlador de joystick.
   - Inicializa el joystick.
4. **Bucle principal para la entrada del joystick:**
   - Un bucle infinito que maneja eventos del joystick.
5. **Manejo de eventos de cambio de posición del joystick:**
   - Verifica si el eje Y del joystick izquierdo se mueve hacia arriba o hacia abajo.
   - Llama a las funciones **mover_up()** o **mover_down()** según el movimiento y las condiciones.
6. **Manejo de eventos de botón del joystick:**
   - Verifica si se presiona el botón A para ejecutar el comando, el botón Y para borrar el archivo seleccionado, o el botón LB para realizar una acción específica.
   - Actualiza la variable **verifica** según la condición.

9. Funciones mover_up y mover_down:

Mueven la selección hacia arriba y hacia abajo en la lista.

```
def mover_up():
    index = listbox_archivos.curselection()
    if index and index[0] > 0:
        index = index[0] - 1
        listbox_archivos.selection_clear(0, tk.END)
        listbox_archivos.selection_set(index)
        listbox_archivos.activate(index)
        listbox_archivos.see(index)
        seleccionar_fila(None)
def mover_down():
    index = listbox_archivos.curselection()
    if index and index[0] < listbox_archivos.size() - 1:
        index = index[0] + 1
        listbox_archivos.selection_clear(0, tk.END)
        listbox_archivos.selection_set(index)
        listbox_archivos.activate(index)
        listbox_archivos.see(index)
        seleccionar_fila(None)
```

1. **Obtiene la selección actual en la lista:**
   - Utiliza **listbox_archivos.curselection()** para obtener el índice de la selección actual en la lista de ROMs.
2. **Verifica condiciones para mover hacia arriba o hacia abajo:**

- Verifica si hay una selección y si el índice permite moverse hacia arriba o hacia abajo.

3. **Actualiza la selección en la lista:**
   - Utiliza **listbox_archivos.selection_clear()** y **listbox_archivos.selection_set()** para actualizar la selección.
   - Utiliza **listbox_archivos.activate()** para activar el nuevo índice seleccionado.
   - Utiliza **listbox_archivos.see()** para hacer visible la selección en la lista.
4. **Llama a seleccionar_fila(None):**
   - Llama a **seleccionar_fila(None)** para actualizar la imagen después de mover la selección.

10. Función seleccionar_primer_archivo:

Selecciona automáticamente el primer archivo en la lista.

```
def seleccionar_primer_archivo(event):
    listbox_archivos.select_set(0)
    listbox_archivos.event_generate("<<ListboxSelect>>")
    seleccionar_fila(None)
```

1. **Selecciona automáticamente el primer archivo en la lista:**
   - Utiliza **listbox_archivos.select_set(0)** para seleccionar el primer elemento de la lista.
   - Utiliza **listbox_archivos.event_generate("<<ListboxSelect>>")** para generar un evento de selección.
2. **Llama a seleccionar_fila(None):**
   - Llama a **seleccionar_fila(None)** para actualizar la imagen después de seleccionar automáticamente el primer archivo.

11. Función ejecutar_comando:

Ejecuta el comando asociado al archivo seleccionado (abre el emulador correspondiente).

```
# Función para ejecutar el comando asociado al archivo seleccionado
def ejecutar_comando(event):
    seleccion = listbox_archivos.curselection()
    if seleccion:
        index = seleccion[0]
        nombre_archivo = listbox_archivos.get(index)
        ruta_completa = os.path.join(os.path.dirname(__file__), "roms", nombre_archivo)

        if nombre_archivo.lower().endswith(".smc"):
            comando = f'~/snes9x-1.60/gtk/build/snes9x-gtk "{ruta_completa}"'
        elif nombre_archivo.lower().endswith(".zip"):
            comando = f'/usr/games/mame "{ruta_completa}"'
```

```
    else:
        # Agrega un manejo para otros tipos de archivos si es necesario
        print(f"Error: Extensión de archivo no admitida para {nombre_archivo}")
        return
    os.system(comando)
```

1. **seleccion = listbox_archivos.curselection()**: Obtiene la selección actual en la lista de ROMs.
2. **if seleccion:**: Verifica si hay una selección.
3. **index = seleccion[0]**: Obtiene el índice del archivo seleccionado en la lista.
4. **nombre_archivo = listbox_archivos.get(index)**: Obtiene el nombre del archivo seleccionado.
5. **ruta_completa = os.path.join(os.path.dirname(__file__), "roms", nombre_archivo)**: Construye la ruta completa del archivo seleccionado.
6. **if nombre_archivo.lower().endswith(".smc"): ...**: Verifica la extensión del archivo y crea el comando correspondiente para ejecutar SNES9x o MAME.
7. **os.system(comando)**: Ejecuta el comando para iniciar el emulador correspondiente.

12. Función mostrar_mensaje_copiando:

Muestra un mensaje temporal indicando que se están copiando nuevas ROMs.

```
# Función para mostrar un mensaje temporal de copia de ROMs nuevas
def mostrar_mensaje_copiando():
    mensaje_copiando = tk.Toplevel()
    mensaje_copiando.title("Copiando ROMs Nuevas")
    ancho_pantalla = mensaje_copiando.winfo_screenwidth()
    altura_pantalla = mensaje_copiando.winfo_screenheight()
    x_pos = 700
    y_pos = 500
    mensaje_copiando.geometry(f"300x100+{x_pos}+{y_pos}")
    mensaje_copiando.attributes("-topmost", True)
    etiqueta_mensaje = tk.Label(mensaje_copiando, text="Copiando ROMs Nuevas", font=("Arial", 14))
    etiqueta_mensaje.pack(pady=20)
    mensaje_copiando.after(5000, mensaje_copiando.destroy)  # Cerrar el mensaje después de 5 segundos
```

1. **mensaje_copiando = tk.Toplevel()**: Crea una nueva ventana emergente.
2. **mensaje_copiando.title("Copiando ROMs Nuevas")**: Establece el título de la ventana emergente.
3. **ancho_pantalla = mensaje_copiando.winfo_screenwidth()**: Obtiene el ancho de la pantalla.
4. **altura_pantalla = mensaje_copiando.winfo_screenheight()**: Obtiene la altura de la pantalla.
5. **x_pos = 700**, **y_pos = 500**: Establece la posición de la ventana emergente en la pantalla.

6. **mensaje_copiando.geometry(f"300x100+{x_pos}+{y_pos}")**: Establece las dimensiones y la posición de la ventana emergente.
7. **mensaje_copiando.attributes("-topmost", True)**: Hace que la ventana emergente esté en la parte superior.
8. **etiqueta_mensaje = tk.Label(mensaje_copiando, text="Copiando ROMs Nuevas", font=("Arial", 14))**: Crea una etiqueta con el mensaje.
9. **etiqueta_mensaje.pack(pady=20)**: Empaqueta la etiqueta en la ventana emergente.
10. **mensaje_copiando.after(5000, mensaje_copiando.destroy)**: Cierra la ventana emergente después de 5 segundos.

13. Función copiar_archivos_usb:

Copia archivos desde una unidad USB a la carpeta de ROMs.

```
# Función para copiar archivos desde una unidad USB a la carpeta de ROMs
def copiar_archivos_usb(ruta_usb, ruta_destino):
    extensiones_permitidas = {".png", ".smc", ".zip"}
    archivos_copiados = []
    for archivo in os.listdir(ruta_usb):
        nombre_archivo, extension = os.path.splitext(archivo)
        if extension.lower() in extensiones_permitidas:
            ruta_origen = os.path.join(ruta_usb, archivo)
            ruta_destino_archivo = os.path.join(ruta_destino, archivo)
            if not os.path.exists(ruta_destino_archivo):
                shutil.copy(ruta_origen, ruta_destino)
                archivos_copiados.append(archivo)
                print(f"Archivo copiado: {archivo}")
            else:
                print(f"El archivo {archivo} ya existe en la carpeta de destino.")
    return archivos_copiados
```

1. **extensiones_permitidas = {".png", ".smc", ".zip"}**: Define las extensiones de archivo permitidas.
2. **archivos_copiados = []**: Inicializa una lista para almacenar los archivos copiados.
3. **for archivo in os.listdir(ruta_usb):**: Itera sobre los archivos en la unidad USB.
4. **nombre_archivo, extension = os.path.splitext(archivo)**: Obtiene el nombre y la extensión del archivo.
5. **if extension.lower() in extensiones_permitidas: ...**: Verifica si la extensión está permitida.
6. **ruta_origen = os.path.join(ruta_usb, archivo)**: Construye la ruta completa del archivo en la unidad USB.
7. **ruta_destino_archivo = os.path.join(ruta_destino, archivo)**: Construye la ruta completa de destino para el archivo.

8. **if not os.path.exists(ruta_destino_archivo): ...**: Verifica si el archivo ya existe en la carpeta de destino.
9. **shutil.copy(ruta_origen, ruta_destino)**: Copia el archivo a la carpeta de destino.
10. **archivos_copiados.append(archivo)**: Agrega el nombre del archivo a la lista de archivos copiados.
11. **print(f"Archivo copiado: {archivo}")**: Imprime un mensaje indicando que el archivo se ha copiado.
12. **else: print(f"El archivo {archivo} ya existe en la carpeta de destino.")**: Imprime un mensaje si el archivo ya existe.

14. Funciones cerrar_snes9x y cerrar_mame:

Intentan cerrar los emuladores SNES9x y MAME.

```
# Funciones para cerrar los emuladores SNES9x y MAME

def cerrar_snes9x():
    try:
        subprocess.run(["killall", "-9", "snes9x-gtk"], check=True)
    except subprocess.CalledProcessError:
        print("Error al intentar cerrar SNES9x")


def cerrar_mame():
    try:
        subprocess.run(["killall", "-9", "mame"], check=True)
    except subprocess.CalledProcessError:
        print("Error al intentar cerrar SNES9x")
```

Ambas funciones intentan cerrar los emuladores SNES9x y MAME enviando la señal SIGTERM (**-9**). Se maneja una excepción en caso de que haya un error al intentar cerrar el emulador.

15. Función monitorear_usb:

Monitorea la conexión de unidades USB y copia ROMs nuevas.

```
# Función para monitorear la conexión de unidades USB y copiar ROMs
def monitorear_usb():
    context = pyudev.Context()
    monitor = pyudev.Monitor.from_netlink(context)
    monitor.filter_by(subsystem='block', device_type='disk')

    for device in iter(monitor.poll, None):
        if device.action == 'add':
            cerrar_snes9x()
            cerrar_mame()
            mostrar_mensaje_copiando()
```

```
    time.sleep(3)
    ruta_usb = os.path.join('/media/fac/KINGSTON')

    archivos_copiados = copiar_archivos_usb(ruta_usb, '/home/fac/Desktop/Proyecto/roms')

    for archivo in archivos_copiados:
        if not archivo.lower().endswith(".png"):
            listbox_archivos.insert(tk.END, archivo)
```

1. **context = pyudev.Context()**: Crea un objeto de contexto de udev.
2. **monitor = pyudev.Monitor.from_netlink(context)**: Crea un monitor udev desde el contexto utilizando netlink.
3. **monitor.filter_by(subsystem='block', device_type='disk')**: Filtra el monitor para eventos relacionados con dispositivos de bloque (como unidades USB).
4. **for device in iter(monitor.poll, None):**: Itera sobre los eventos del monitor.
5. **if device.action == 'add':**: Verifica si se ha conectado una unidad USB.
6. **cerrar_snes9x()**, **cerrar_mame()**: Intenta cerrar los emuladores SNES9x y MAME.
7. **mostrar_mensaje_copiando()**: Muestra un mensaje temporal de copia de ROMs.
8. **time.sleep(3)**: Espera 3 segundos para asegurarse de que la unidad USB esté completamente montada.
9. **ruta_usb = os.path.join('/media/fac/KINGSTON')**: Establece la ruta de la unidad USB.
10. **archivos_copiados = copiar_archivos_usb(ruta_usb, '/home/fac/Desktop/Proyecto/roms')**: Copia los archivos desde la unidad USB a la carpeta de ROMs.
11. **for archivo in archivos_copiados: ...**: Itera sobre los archivos copiados y los agrega a la lista de ROMs en la interfaz gráfica.

```
# Creación de la ventana principal de la aplicación
ventana = tk.Tk()
ventana.title("FAC Retro Realm")
ventana.attributes('-fullscreen', True)
# Creación de fuentes personalizadas y carga de imágenes de fondo y banner
fuente_personalizada = font.Font(family="Helvetica", size=24, weight="bold", slant="italic")
ruta_fondo = os.path.join(os.path.dirname(__file__), "fondo.png")
imagen_fondo = Image.open(ruta_fondo)
ancho_ventana = ventana.winfo_screenwidth()
altura_ventana = ventana.winfo_screenheight()
imagen_fondo = imagen_fondo.resize((ancho_ventana, altura_ventana), Image.LANCZOS)
imagen_fondo_tk = ImageTk.PhotoImage(imagen_fondo)
# Creación de widgets (etiquetas, listbox, botones) y configuración de la interfaz
```

```
etiqueta_fondo = tk.Label(ventana, image=imagen_fondo_tk, borderwidth=0)
etiqueta_fondo.place(relx=0, rely=0, relwidth=1, relheight=1)
ruta_banner = os.path.join(os.path.dirname(__file__), "banner.png")
imagen_banner = Image.open(ruta_banner)
imagen_banner = imagen_banner.resize((ancho_ventana, int(altura_ventana * 0.2)), Image.LANCZOS)
imagen_banner_tk = ImageTk.PhotoImage(imagen_banner)
etiqueta_banner = tk.Label(ventana, image=imagen_banner_tk, borderwidth=0)
etiqueta_banner.pack(side="top", fill="x")
canvas_lista = tk.Canvas(ventana, width=700, height=altura_ventana, bg="white")
canvas_lista.pack(side="left", fill="both", expand=False, padx=20, pady=20)
nombres_archivos = cargar_nombres_archivos()
listbox_archivos = tk.Listbox(canvas_lista, font=("Arial", 14), selectbackground="gray", selectmode=tk.SINGLE,
bg="white", width=40)
for nombre_archivo in nombres_archivos:
    listbox_archivos.insert(tk.END, nombre_archivo)
listbox_archivos.pack(fill="both", expand=True)
listbox_archivos.bind("<ButtonRelease-1>", seleccionar_fila)
etiqueta_imagen = tk.Label(ventana, bg="white")
etiqueta_imagen.pack(side="left", padx=0)
etiqueta_imagen_sistema = tk.Label(ventana, bg="white")
etiqueta_imagen_sistema.pack(side="left", padx=0)
ancho_pantalla = ventana.winfo_screenwidth()
altura_pantalla = ventana.winfo_screenheight()
ancho_imagen = imagen_fondo.width
altura_imagen = imagen_fondo.height
posicion_vertical = (altura_pantalla - altura_imagen) // 2 + 540
etiqueta_imagen.place(relx=0.5, rely=posicion_vertical/altura_pantalla, anchor="center")
etiqueta_imagen_sistema.place(relx=0.8, rely=posicion_vertical/altura_pantalla, anchor="center")
ruta_boton = os.path.join(os.path.dirname(__file__), "boton.png")
imagen_boton = Image.open(ruta_boton)
imagen_boton_tk = ImageTk.PhotoImage(imagen_boton)

ancho_boton = imagen_boton.width
altura_boton = imagen_boton.height
posicion_vertical_boton = (altura_pantalla - altura_boton) // 2 - 100
etiqueta_boton = tk.Label(ventana, image=imagen_boton_tk, borderwidth=0)
etiqueta_boton.pack(side="bottom", pady=20)
etiqueta_boton.place(relx=0.5, rely=posicion_vertical_boton/altura_pantalla, anchor="center")
# Inicio de hilos para la entrada del joystick y el monitoreo de USB
hilo_joystick = threading.Thread(target=joystick_input)
hilo_joystick.daemon = True
hilo_joystick.start()
```

```
hilo_usb = threading.Thread(target=monitorear_usb)
hilo_usb.daemon = True
hilo_usb.start()
# Llamada a la función para seleccionar automáticamente el primer archivo
seleccionar_primer_archivo(None)
# Inicio del bucle principal de la interfaz gráfica
ventana.mainloop()
```

Creación de la ventana principal de la aplicación y configuración de la interfaz gráfica.

Utiliza Tkinter para crear la ventana, etiquetas, listbox, y otros elementos de la interfaz.

Inicia hilos para la entrada del joystick y el monitoreo de USB.

Inicio del bucle principal de la interfaz gráfica (ventana.mainloop()):

Inicia el bucle principal de la interfaz, manteniendo la aplicación en ejecución.

Este script parece ser parte de una aplicación para gestionar y jugar ROMs de juegos retro en emuladores.