**Fundamentos de sistemas embebidos. Gpo. 3. Sem. 2025-2**
**313123762 – Cortés Hernández José César**
**Documentación de proyecto final. Retroconsola ccjpmmGaming**
**Énfasis en la detección de periféricos USB**
**20/05/25**

## 1. Objetivo

Desarrollar una retroconsola funcional y optimizada en una Raspberry Pi, utilizando Python como lenguaje principal sobre una instalación limpia de Raspberry Pi OS Lite, que permita emular sistemas clásicos de videojuegos con una interfaz amigable, controles configurables y cumpliendo con las siguientes características:

- **Copia automática de ROMs** desde una USB al momento de conectarla, con verificación de formatos soportados y organización en directorios por sistema.
- **Menú de búsqueda de ROMs** con un método de categorización por consola.
- **Completo control mediante gamepad/joystick**, incluyendo:
  - Navegación en la interfaz.
  - Hotkeys para funciones rápidas (detener emulación, apagar consola, acceder al menú de búsqueda).
- **Arranque automático** de la consola al iniciar la Raspberry Pi.
- **Interfaz gráfica intuitiva** con:
  - Listado de juegos por consola, mostrando una imagen para identificar el juego.

## 2. Introducción

En los últimos años, el interés por los videojuegos retro ha experimentado un notable resurgimiento, impulsado por la nostalgia y el deseo de preservar la historia de los videojuegos. Clásicos de consolas como NES, SNES y GBA han recobrado relevancia, atrayendo tanto a quienes vivieron su época dorada como a nuevas generaciones curiosas por conocer sus raíces. Sin embargo, acceder a hardware original puede ser costoso, difícil de mantener e incluso inaccesible, debido a su limitada disponibilidad y obsolescencia. Esto ha llevado a muchos entusiastas a recurrir a soluciones de emulación como una alternativa práctica y económica.

En este contexto, las placas de bajo costo, como la Raspberry Pi, se han convertido en una plataforma ideal para este propósito, gracias a su versatilidad, potencia suficiente para emular múltiples consolas clásicas, y su bajo consumo energético. Además, su comunidad activa y amplia documentación facilitan el desarrollo de proyectos personalizados y escalables.

Este proyecto busca desarrollar una retroconsola funcional y optimizada utilizando una Raspberry Pi con Raspberry Pi OS Lite como sistema base, aprovechando la eficiencia de Python como lenguaje principal. El objetivo es crear una solución accesible, fácil de configurar y con una interfaz intuitiva que permita a los usuarios disfrutar de sus juegos retro favoritos sin complicaciones técnicas.

Además de ser una fuente de entretenimiento personalizada, esta retroconsola funcionará como una plataforma educativa para explorar conceptos de emulación, desarrollo de interfaces gráficas con Pygame, automatización de procesos mediante scripts y administración de archivos en sistemas embebidos. El resultado será una solución integral, documentada, estable y lista para ser utilizada tanto por aficionados como por usuarios casuales que deseen revivir la era dorada de los videojuegos desde un enfoque moderno y accesible.

## 3. Marco teórico

### 3.1 Sistemas embebidos

Un sistema embebido es un dispositivo electrónico que incorpora un computador programable (como un microcontrolador o microprocesador), pero no se considera un computador de propósito general. Está diseñado para realizar tareas específicas dentro de un sistema mayor, de forma eficiente y muchas veces sin ser visible para el usuario.

Estos sistemas se componen de tres elementos principales:

1. **Hardware**,
2. **Software embebido** (aplicación principal),
3. **Sistema operativo**, el cual frecuentemente es de tiempo real para garantizar respuestas rápidas y precisas.

El software se diseña con importantes restricciones, como uso mínimo de memoria, bajo consumo de energía y capacidad limitada de procesamiento. Un ejemplo típico de arquitectura embebida incluye procesadores RISC, memoria FLASH, RAM limitada, y puertos de comunicación como Ethernet o USB.

**Características clave:**

- **Funcionamiento específico**: ejecuta siempre la misma tarea o conjunto reducido de tareas.
- **Fuertes limitaciones de diseño**: bajo costo, tamaño reducido, alta eficiencia energética y buen desempeño.
- **Reactividad y tiempo real**: debe responder a eventos del entorno en tiempos determinados, como en el caso de los sistemas de control automotriz.

### 3.2 Raspberry Pi como sistema embebido

La Raspberry Pi es un ordenador de placa única (Single Board Computer, SBC) sin partes móviles, diseñado principalmente como plataforma para la enseñanza de programación y el control de periféricos a bajo nivel. Existen varias versiones comerciales basadas en un SoC de la misma familia, con diferencias en características técnicas. A pesar de su apariencia modesta y menor rendimiento comparado con PCs o laptops, es suficientemente potente para aplicaciones complejas, incluso en robótica o exploración espacial.

Como sistema embebido, la Raspberry Pi se diferencia de los microcontroladores tradicionales (como Arduino) en que es un ordenador completo con procesador ARM, memoria RAM variable (de 256 MB a 8 GB) y almacenamiento mediante tarjetas SD o microSD. Esto le permite ejecutar sistemas operativos completos, principalmente Raspbian (una versión adaptada de Linux Debian), aunque también otros sistemas operativos especializados para usos concretos. Esta flexibilidad facilita la experimentación, desarrollo y aprendizaje, permitiendo preparar múltiples imágenes con diferentes configuraciones y software.

En el contexto de sistemas embebidos, la Raspberry Pi es una plataforma más robusta y versátil que los microcontroladores clásicos, orientada a aplicaciones que requieren mayor capacidad de cómputo y un sistema operativo que brinde soporte a librerías y paquetes complejos. Sin embargo, para aplicaciones de tiempo real estrictas o con recursos muy limitados, se suelen preferir microcontroladores más simples.

### 3.3 Python para el desarrollo de videojuegos

Python es un lenguaje de programación de alto nivel, interpretado, de propósito general, moderno y accesible, que junto con la biblioteca **PyGame**, es una excelente opción para el desarrollo de videojuegos, especialmente en plataformas como la Raspberry Pi. PyGame es un módulo de Python que facilita la creación de juegos en 2D mediante el manejo sencillo de gráficos, **sprites**, sonidos y eventos de entrada como teclado y joystick, lo que permite a los desarrolladores centrarse en la lógica del juego sin complicaciones excesivas.

En el contexto de la Raspberry Pi, PyGame viene preinstalado en la versión oficial de Raspbian, lo que simplifica el inicio en el desarrollo de juegos. La estructura básica de un juego con PyGame incluye la inicialización del juego, un bucle principal donde se gestionan eventos, se actualiza la lógica y se redibuja la pantalla, lo que es ideal para aprender conceptos fundamentales de programación de videojuegos

## 4. Materiales

Los materiales para poder instalar esta retroconsola en una Raspberry Pi con una instalación limpia de Raspios Lite son los siguiente.

- Raspberry Pi 4 (Recomendablemente de 4 – 8 GB para mejor rendimiento durante la emulación)
- Conexión a internet (Para la auto instalación)
- Control de Xbox Series S/X (Es posible configurar otro control, el cómo hacerlo se especifica en la nota adjunta en la sección 5.7)
- Bocinas externas con conexión Jack 3.5 (Para la salida de audio de la consola)
- Monitor con entrada HDMI (O adaptador para poder obtener imagen desde conexión micro HDMI)

## 5. Configuración e instalación en Raspberri PI 4 con Raspios Lite

Para proceder con la instalación de la retroconsola hay que ejecutar la siguiente línea como usuario ordinario:

```
[1] $ sudo bash -c "$(curl -fsSL \
      https://raw.githubusercontent.com/JuanPer03/ccjpmmGaming/Instalacion/instalacion.sh)"
```

Lo que se hace es ejecutar un script que se encargara de hacer la configuración necesaria para que la retroconsola funcione sin problemas (proceso que dura de 10 a 15 minutos), las acciones que se ejecutaran automáticamente son:

a) Actualización del sistema base.
b) Instalación de Git y clonación de la rama *Instalacion* del repositorio en GitHub del proyecto. (enalce en la sección 7)
c) Instalación de dependencias básicas de Python incluyendo Pygame y PYUDEV.
d) Instalación del emulador Mednafen y dependencias adicionales para Raspberry Pi
e) Creación de los directorios para la descompresión de los archivos de la retroconsola.
f) Habilitación de autoarranque de la retroconsola al encender la Raspberry Pi.
g) Aplicación de configuración de emulador para el control de Xbox Series S/X.

A continuación, se explicará cada sección del script *instalación.sh* a modo de instructivo para hacer la instalación manual en caso de no querer hacer uso del script de auto instalación.

### 5.1 Actualización del sistema base

Para comenzar con la configuración de la Raspberry Pi para que ejecute sin problema la retroconsola ccjpmmGaming, como primer paso es necesario actualizar el sistema base, esto ejecutando la siguiente línea:

```
$ sudo apt update && sudo apt upgrade -y
```

Lo que se hace es actualizar la lista local de paquetes disponibles en los repositorios configurados dentro de Raspios Lite, descargando la información más reciente sobre versiones de software y actualizaciones, una vez que se haya actualizado la lista de paquetes, el sistema instalara todas las actualizaciones disponibles para los paquetes ya instalados.

### 5.2 Instalación de Git y clonación de la información del repositorio de GitHub

Una vez actualizado el sistema, es necesario descargar la información correspondiente de la retroconsola, esta información está dentro de un repositorio de GitHub, entonces se tiene que instalar Git y clonar la rama Instalacion del repositorio mencionado anteriormente de la siguiente manera:

```
$ sudo apt install git -y
$ sudo git clone -b Instalacion --single-branch https://github.com/JuanPer03/ccjpmmGaming.git
```

En este punto se esta instalando Git, una herramienta de control de versiones que permite clonar, actualizar y gestionar repositorios de código fuente, especialmente desde plataformas como GitHub, esto con el fin de poder descargar una copia de la rama *Instalacion*, la cual contiene:

- Archivo comprimido con la estructura de las carpetas y archivos necesarios para que la retroconsola funcione.
- Configuración de Mednafen para control de Xbox Series S/X.
- Script de instalación automática.
- Archivos comprimidos con ROMs extra a las 15 que debería de tener por defecto.

### 5.3 Instalación de Python 3 y dependencias básicas, Pygame y dependencias básicas y PYUDEV.

Para la configurar correctamente el entorno de Python (lenguaje de programación sobre el cual será ejecutado el código principal), es necesario descargar las dependencias necesarias para que el código principal no cause errores de ejecución de la siguiente manera:

```
$ sudo apt install -y python3 python3-pip python3-dev
$ sudo apt install -y libsdl2-dev libsdl2-mixer-dev libsdl2-image-dev libsdl2-ttf-dev
$ sudo apt install -y python3 python3-pygame
$ sudo apt install -y python3 python3-pyudev
```

- **Python3.** Interprete de Python para ejecutar el programa.
- **SDL2.** Biblioteca multiplataforma que provee acceso a bajo nivel a hardware de audio, gráficos, teclado, ratón y joystick.
- **Pygame.** Conjunto de módulos basados en SDL para la creación de videojuegos 2D.
- **Pyudev.** Biblioteca que permite detectar y gestionar eventos de dispositivos USB y harware en Linux.

La inclusión de estas dependencias es para poder tener control de la detección de conexión y desconexión de memorias USB y del control que se está usando para navegar por el sistema, así mismo, serán de gran importancia al momento de desarrollar las interfaces graficas para navegar entre los diferentes menús de la retroconsola.

## 5.4 Instalación de Mednafen

La retroconsola toma como emulador principal a Mednafen[1] para ejecutar juegos de las consolas Game Boy Advanced, Nintendo Entertaiment System y Super Nintendo Entertaiment System, el programa principal se encarga de llamarlo al momento de seleccionar la ROM del juego que se desea jugar y lo ejecuta, se instala de la siguiente manera:

```
$ sudo apt install -y mednafen
```

## 5.5 Estructura de carpetas para la retroconsola

Hasta este punto ya está instalado todo lo necesario para el correcto funcionamiento de la consola, pero hace falta construir la estructura de las rutas para que el programa principal busque todos los recursos necesarios para la correcta ejecución, para esto se tienen que crear las siguientes carpetas:

```
$ mkdir -p /home/ccjpmmGaming
$ mkdir -p /home/ccjpmmGaming/usb
$ mkdir -p ~/.mednafen
```

- ***ccjpmmGaming.*** Es la carpeta raíz de la retroconsola.
- ***ccjpmmGaming/usb.*** Es la carpeta donde se estará montando automáticamente la memoria USB para realizar la copia de las ROMs.
- ***~/.mednafen.*** Es la carpeta donde se almacena la configuración del emulador (se crea en cuanto se ejecuta por primera vez el juego, pero se hace la creación antes de hacerlo para asegurar la copia del archivo de configuración de la sección 5.7)

Una vez creados los directorios, se puede descomprimir el contenido de los archivos .zip que se descargaron del contenedor de GitHub en la sección 5.2 de la siguiente manera:

```
$ unzip ccjpmmGaming/Retroconsole.zip -d /home/ccjpmmGaming
$ unzip ccjpmmGaming/RomsGBA1.zip -d /home/ccjpmmGaming/Retroconsole/roms/GBA
$ unzip ccjpmmGaming/RomsGBA2.zip -d /home/ccjpmmGaming/Retroconsole/roms/GBA
$ unzip ccjpmmGaming/RomsNES.zip -d /home/ccjpmmGaming/Retroconsole/roms/NES
$ unzip ccjpmmGaming/RomsSNES.zip -d /home/ccjpmmGaming/Retroconsole/roms/SNES
```

- **Retroconsole.zip** contiene las carpetas donde se encuentran las 15 ROMs por defecto, las imágenes que fungirán como caratulas de las ROMs, las imágenes donde se explican el mapeo de controles, logo personalizado y la carpeta donde está el código principal
- **RosGBA1.zip, RomsGBA2.zip, RomsNES.zip y RomsSNES.zip** son archivos que contienes ROMs extra a las 15 que vienen por defecto.

Esto asegurara la correcta ejecución del programa principal de la retroconsola, ya que se basa en estas direcciones para obtener los archivos que utilizara durante la ejecución.

[1] Mednafen es un emulador multi-sistema de código abierto que funciona mediante línea de comandos y utiliza OpenGL y SDL para la emulación gráfica y de audio
Referencia. *Medafen - Multi-system Emulator*. (s. f.). https://mednafen.github.io/

### 5.6 Configuración de autoarranque de la retroconsola

Para configurar el arranque automático de la retroconsola es necesario modificar Cron[2] para que ejecute el código principal de la retroconsola en cuanto la Raspberry Pi se reinicie, de la siguiente manera:

```
$ chmod +x /home/ccjpmmGaming/Retroconsole/code/Code.py
$ touch "/home/ccjpmmGaming/log.txt "
$ (crontab -l 2>/dev/null; echo "@reboot python3 /home/ccjpmmGaming/Retroconsole/code/Code.py
  > /home/ccjpmmGaming/log.txt 2>&1") | crontab -
$ sudo systemctl enable cron
$ sudo systemctl start cron
```

Lo que se está haciendo es:

1. Hacer que el programa principal de la retroconsola sea ejecutable.
2. Crear un archivo vacío que guardara los logs (salida y errores) cuando se ejecute el programa.
3. Agregamos al final del archivo de configuración de Cron la instrucción para que cuando se reinicie la Raspberry Pi se ejecute el programa principal.
4. Habilitar e iniciar el servicio de Cron

Una vez configurado el inicio automático de la retroconsola, hay que deshabilitar el volcado de mensajes en la terminal para un arranque más limpio del sistema, para esto, es necesario modificar el archivo de configuración de arranque */boot/firmware/cmdline.txt* con ayuda de un editor de texto (nano por ejemplo), agregando el siguiente contenido al final de la primera línea que se encuentra en este archivo:

```
quiet loglevel=0 logo.nologo fsck.mode=skip
```

Esto deshabilitara el volcado de mensajes al momento de iniciar el sistema, haciendo que lo primero que se vea al encender la pantalla sea el programa de la retráncanosla.

### 5.7 Configuración de Mednafen

Finalmente se aplicará la configuración preestablecida de Mednafen para un control de Xbox Series S/X, de la siguiente manera:

```
$ sudo -u <NombreDeUsuario> mednafen & sleep 10 && kill $!
$ sudo cp ccjpmmGaming/mednafen.cfg ~/.mednafen/mednafen.cfg
```

Esto ejecutara durante 10 segundos Mednafen para generar el archivo de configuración del emulador, hay que modificar `<NombreDeUsuario>` por el nombre de tu usuario y posterior a esto se copia el archivo que se descargo al momento de clonar la rama *Instalacion* del repositorio en GitHub del proyecto en la sección 5.2, el cual contiene la configuración creada para el control de Xbox Series S/X.

**Nota. En caso de querer mapear un control diferente es necesario conectar un teclado por medio de algún puerto USB de la Raspberry Pi durante la emulación y presionar simultáneamente Alt+Ctrl+1 para comenzar con el mapeo del nuevo control, si se desea conectar un segundo control es necesario presionar durante la emulación Alt+Ctrl+2 para que el emulador configure el control.**

[2] Cron es un servicio que se incia automáticamente al arrancar el sistema y funciona como un administrador de tareas programadas en segundo plano, permite ejecutar comandos o scripts en momentos específicos o de forma periódica.

Referencia. Fromaget, P. (s. f.). *¿Cómo Programar una Tarea en una Raspberry Pi? – RaspberryTips*. https://raspberrytips.es/programar-tarea-raspberry-pi/

## 6. Descripción e implementación de los módulos de programación

### 6.1 Módulos relacionados con la detección de USB y conexión de control

#### 6.1.1 Funciones gestion USB

**check_existing_usb()**

La función revisa si hay un USB conectado al iniciar el sistema, retorna True si se encuentra alguna y la monta. Si no lo logra manda un error.

```
[1]     print(f"Error al verificar USB existente: {e}")
```

**check_and_mount_usb()**

La función verifica y monta la USB en el directorio especifico, sino existe crea el directorio en la ruta y se procede a montarla en alguna de las direcciones más comunes como lo son /sda1, /dev, /sdc1.

```
[1]     for device in ['/dev/sda1', '/dev/sdb1', '/dev/sdc1']:
[2]           os.makedirs(USB_MOUNT_DIR)
[3]           if os.path.exists(device):
[4]                 subprocess.run(['sudo', 'mount', device, USB_MOUNT_DIR],check=True)
[5]                       return True
```

Una vez hecho el montaje se procede con la verificación.

**unmount_usb()**

Para realizar el desmonte de la USB de forma segura, se utiliza el comando *unmount*, en caso de algún error al hacer la operación habrá un mensaje.

```
[1]    if os.path.ismount(USB_MOUNT_DIR):
[2]               subprocess.run(['sudo', 'umount', USB_MOUNT_DIR], check=True)
[3]               print("Dispositivo USB desmontado")
[4]    except Exception as e:
[5]                     print(f"Error al desmontar USB: {e}")
```

Al ejecutar esa línea, aseguramos que no haya errores por extracción del dispositivo.

**copy_roms_from_usb()**

Esta función copia las ROMS desde la USB leída previamente, hacia los directorios donde se alojarán dependiendo la extensión de estos archivos. También se tiene la verificación en caso de que se encuentre una ROM.

Se recorre el contenido del directorio por defecto */home/ccjpmmGaming/usb*, mediante un ciclo que comprueba los archivos. En caso de que no esté montada la USB, se procede a mandar un mensaje de error.

```
[1]    if not os.path.isdir(USB_MOUNT_DIR):
[2]           return copied_files, False
```

Una vez dentro del directorio, se comprueba si hay archivos con extensiones compatibles con el emulador (gba, nes, smc, sfc). Si es el caso, se procede a definir el directorio de destino que tendrán dependiendo su extensión.

```
[1]    for ext in USB_ROM_DIRS.keys():
[2]           if filename.lower().endswith(ext):
[3]                  found_roms = True
[4]                        dest_dir = USB_ROM_DIRS[ext]
```

Si no existe un directorio en la ruta lo crea con la siguiente comprobación.

```
[1]    if not os.path.exists(dest_dir):
[2]                           os.makedirs(dest_dir)
```

Se verifica si el archivo es nuevo, en caso de que este ya se encuentre en la lista del directorio destino se omite y se pasa el valor a la variable.

```
[1]    should_copy = False
```

Sino existen se cambia el valor de la variable a *True* para proceder a la copia. En caso contrario se asignan valores de fecha a las variables *src_mtime* y *dst_mtime* para determinar si se ha modificado un archivo.

```
[1]    if not os.path.exists(dest_path):
[2]                          should_copy = True
[3]    else:
[4]           src_mtime = os.path.getmtime(filepath)
[5]                             dst_mtime = os.path.getmtime(dest_path)
```

Se comparan los valores para determinar si se copia el archivo a el directorio.

```
 [1]   if src_mtime > dst_mtime:
 [2]          should_copy = True
```

Una vez que la variable *should_copy*, tenga valor positivo, se procede a hacer el proceso de copiar.

```
[1]    if should_copy:
[2]                           shutil.copy2(filepath, dest_path)
[3]                           copied_files[ext].append(filename)
[4]                     print(f"Copiada {filename} a {dest_dir}")
```

**setup_usb_monitor()**

Esta función se encarga de configurar el monitor de eventos USB utilizando la librería *pyudev*, para lograrlo se requiere crear un contexto del sistema con la función *context()*, esto hace que pueda interactuar con dispositivos del sistema. También se hace uso del monitor de eventos que escucha la conexión y desconexión de dispositivos a través de *netlink*.

```
[1]    context = pyudev.Context()
[2]    monitor = pyudev.Monitor.from_netlink(context)
```

Se aplica un filtro para que el monitor escuche eventos relacionados con bloques, en este caso se requiere que sea de tipo *disk*.

```
[1]    monitor.filter_by(subsystem='block', device_type='disk')
```

De esta manera se podría detectar cuando se conecte y desconecte la USB.

**usb_event_handler(game_state)**

La función maneja los eventos de conexión y desconexión de dispositivos usb en tiempo real, usando un hilo, para que no interrumpa el bloque de la aplicación principal.

Realizando llamadas a las funciones *check_existing_usb()*, *check_and_mount_usb()*, para verificar la conexión de dispositivo USB, es posible seguir con el flujo del programa para la posteriormente seguir con el programa.

```
[1]    for device in iter(monitor.poll, None):
[2]           if device.action == 'add':
[3]                  print("Dispositivo USB conectado")
```

**start_usb_monitor(game_state)**

La función inicializa el hilo que monitorea la conexión USB, si llegara a fallar retorna la referencia al hilo.

```
[1]    USB_thread = Thread(target=usb_event_handler, args=(game_state,), daemon=True)
[2]    usb_thread.start()
```

**show_copy_confirmation(screen, copied_files)**

Muestra una notificación cuando los archivos son copiados desde la USB y espera la confirmación del usuario para continuar. Utiliza como argumentos el estado de la simulación y los archivos copiados.

```
[1]    dialog_rect = pygame.Rect(70, 120, 500, 240)
[2]    pygame.draw.rect(overlay, COLOR_BG, dialog_rect, border_radius=10)
[3]    pygame.draw.rect(overlay, COLOR_HIGHLIGHT, dialog_rect, 3, border_radius=10)
```

Se muestra la lista de archivos copiados

```
[1]    msg = font_small.render(f"Se copiaron {total_copied} ROMs:", True, COLOR_TEXT)
[2]    overlay.blit(msg, (320 - msg.get_width()//2, 200))
```

Para la confirmación tenemos un ciclo que espera el evento JOYBUTTONDOWN y button, para cambiar el valor de la variable *waiting* y seguir con el programa.

```
[1]    if event.type == pygame.JOYBUTTONDOWN and event.button == 0:
[2]           waiting = False
```

**show_connect_controller(require_button_press=True)**

La función muestra la pantalla de conexión del control, espera hasta que sea conectado y presione un botón para continuar, retorna el mensaje de joystick conectado.

```
[1]    joystick_count = pygame.joystick.get_count()
```

Muestrando la notificación de control conectado

```
[1]    title = font_large.render("Control Conectado", True, COLOR_HIGHLIGHT)
[2]          instruction = font_small.render("Presiona el botón A para continuar", True,
COLOR_TEXT)
```

Espera al usuario a presionar el botón.

```
[1]    for event in pygame.event.get():
[2]                if event.type == pygame.JOYBUTTONDOWN and joystick_connected and
event.button == 0:
[3]                  button_pressed = True
[4]                        return pygame.joystick.Joystick(0)
```

**init_inputs()**

Esta función inicializa los sistemas de entrada y salida, también muestra la pantalla de conexión del control.

```
 [1]   pygame.init()
 [2]   pygame.joystick.init()
 [3]   return show_connect_controller()
```

### 6.2 Módulos relacionados con el inicio y termino de la emulación del juego

En este módulo se hace referencia a la elección del emulador Mednafen, el cual soporta juegos desde los 8 a los 32 bits mediante una arquitectura unificada. Por estas características fue nuestra elección para utilizarlo como el emulador para la consola retro, montada en la Raspberry Pi4.

Para lograr una integración correcta del emulador y el sistema desarrollado, se utilizó algunas características como lo es el mapeo de controles contextual, la compatibilidad ampliada y la gestión del rendimiento mediante el escalado de hardware.

El diseño está pensado para priorizar el inicio controlado, visualización intuitiva, monitoreo en tiempo real y gestión de errores los cuales son los componentes clave de este módulo. Se comienza con el proceso de emulación, siguiendo el flujo de la operación.

Tomando en consideración cada parte de la emulación durante la ejecución del juego, se tiene la verificación de conexión del control, la detección de comando de apagado y monitoreo general, son algunas de las funcionalidades que han sido desarrolladas.

Para una mejor adecuación a los controles se ha designado una imagen de referencia con la configuración equivalente a los controles originales a partir de un mando de Xbox series X/S, para las emulaciones.

### 6.3 Módulos relacionados con la creación de la interfaz gráfica (Menú principal)

La visualización es de las partes más importantes en este proyecto, se desarrolló una pantalla de inicio, caratulas y visualización de estas, organización de los archivos y navegación entre menús para seleccionar el juego.

Para el desarrollo de la pantalla de inicio se implementó la visualización en una ventana que se desplegara y es donde se emularán los juegos. Asimismo, se ha habilitado el uso del audio y la carga del archivo a reproducir. Esto se reproducirá al arrancar el sistema.

Para el menú se ha dividido en tres partes, títulos, área de títulos (ROMS disponibles) y área de controles. Cada una de ellas con información concisa y visual. Para mostrar las caratulas se hace uso de la búsqueda en la carpeta donde se aloja y luego se muestra en las coordenadas definidas previamente. El menú de búsqueda hace uso del teclado virtual, con sus caracteres definidos en una estructura.

Tomando en cuenta el renderizado automático, se tomó en cuenta el diseño de como se muestra este teclado, así como la distribución de cada elemento. Para el módulo de navegación se debe interpretar los botones presionados durante la ejecución del programa y mapearlo del mando a la pantalla.

Para abrir el menú de búsqueda se implementó una combinación de botones para abrirlo desde la pantalla en la que se esté actualmente. También se muestra una vez ingresado lo que se busca, la lista de ROMS que filtra la búsqueda. Se cuenta con una función de mapeo por extensión el cual se encarga de clasificar las ROMS por plataforma para mostrar los juegos disponibles dentro del menú principal.

## 7. Resultados obtenidos

La consola en funcionamiento se muestra en un video que se subió a la plataforma de YouTube, al cual se puede acceder mediante el siguiente enlace.

https://youtu.be/znDL1qFbyX8

De igual forma, el enlace estará adjunto junto el archivo comprimido de toda la documentación del proyecto y el siguiente enlace del repositorio de GitHub del proyecto.

https://github.com/JuanPer03/ccjpmmGaming

Donde se encuentran los siguientes archivos:

- Código principal de la consola.
- Script de auto instalación.
- Archivo README.
- Archivo comprimido de la documentación del proyecto, el cual incluye todos los tutoriales con énfasis de desarrollo especifico.

## 8. Conclusiones

La implementación de los módulos de programación del sistema con un enfoque especializado y bien estructurado en la gestión de dispositivos USB, la interfaz gráfica y la integración del emulador. Se automatizó la detección, montaje y desmontaje de unidades USB, asegurando un flujo continuo y seguro en la transferencia de archivos. La función de copiado clasifica las ROMs según su extensión, copiando solo archivos nuevos o modificados, lo que optimiza el uso del almacenamiento. Además, el sistema incorpora un monitoreo en tiempo real de eventos USB mediante hilos y la librería pyudev, lo que permite mantener una respuesta dinámica sin afectar el rendimiento general. La experiencia de usuario está ambiciosamente diseñada, proporcionando notificaciones en pantalla para la conexión de controles y la confirmación de copias, mejorando así la experiencia de uso. Se eligió el emulador Mednafen por su compatibilidad con múltiples plataformas retro y su arquitectura unificada, permitiendo una integración eficiente con la Raspberry Pi 4. Finalmente, la interfaz gráfica está organizada de forma clara y funcional, con menús intuitivos, un teclado virtual y una clasificación automática de ROMs, todo optimizado para hardware específico como mandos de Xbox en el caso del teclado virtual y la Raspberry Pi para una consola retro que hemos desarrollado.

## 9. Cuestionario

a) ¿Qué función se encarga de monitorear en tiempo real la conexión/desconexión de dispositivos USB y cómo maneja la copia automática de ROMs cuando se detecta un USB?
b) Describe el proceso que sigue la función launch_game() desde que se selecciona un juego hasta que comienza la emulación, incluyendo cómo se muestran los controles mapeados.
c) ¿Cómo está estructurado el sistema de búsqueda de juegos en el código y qué criterios utiliza para organizar y mostrar los resultados al usuario?
d) ¿Cómo se implementa el teclado virtual para la búsqueda de ROMs en la función *show_search_keyboard()* y qué consideraciones de diseño se tuvieron en cuenta para su interacción mediante controles de juego?
e) ¿Qué mecanismos utiliza el sistema para gestionar la ejecución concurrente del emulador (con *emulator_process()*), el monitoreo de USB en segundo plano, y cómo se coordinan estas operaciones con el hilo principal de la interfaz?

## 10. Referencias

[1] A. Perez, D., A. (2009). *Sistemas embebidos y sistemas operativos embebidos*. Centro de Investigación en Comunicación y Redes (CICORE).

[2] Salcedo, M. (2015). Minicomputador educacional de bajo costo Raspberry Pi: primera parte. *REVISTA ETHOS VENEZOLANA*, *7*(1). https://biblat.unam.mx/hevila/RevistaEthosvenezolana/2015/vol7/no1/2.pdf

[3] Kelly, S. (2019). *Python, PyGame, and Raspberry Pi Game Development*.