Fundamentos de sistemas embebidos. Gpo. 3. Sem. 2025-2
317355723 – Peralta Rodríguez Juan Manuel
Énfasis en la creación de la interfaz grafica
Documentación de proyecto final. Retroconsola ccjpmmGaming
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 esta 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

[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

El sistema implementa una gestión integral para la detección y manejo de dispositivos USB y controles de la retroconsola de emulación. En cuanto a la detección y gestión de USB, el sistema realiza un montaje automático de dispositivos comunes al iniciar, verificando la presencia de unidades USB en */dev/sda1* o */dev/sdb1* mediante la función *check_existing_usb()*. Para ello, crea los directorios necesarios para el montaje y maneja posibles errores de conexión. Además, utiliza la biblioteca *Pyudev* para monitorear en tiempo real la conexión y desconexión de dispositivos USB a través de un hilo separado que ejecuta *setup_usb_monitor()*, permitiendo una detección asíncrona sin bloquear la ejecución principal. En cuanto a la gestión de archivos, el sistema identifica archivos de juegos por sus extensiones (*.gba* o *.nes*) y copia estos de forma inteligente a directorios específicos, sincronizando únicamente los archivos nuevos o modificados comparando las fechas de modificación. Durante este proceso, si un emulador está en ejecución, se detiene para evitar conflictos y se notifica al usuario sobre los archivos copiados mediante mensajes de confirmación. Finalmente, tras completar la copia, el dispositivo USB se desmonta de manera segura para evitar pérdidas de datos.

Respecto al sistema de controles, se presenta una interfaz gráfica que guía al usuario para conectar el control, utilizando la biblioteca *Pygame* para inicializar y detectar dispositivos de entrada. El sistema espera la interacción del usuario, específicamente la pulsación del botón A, para confirmar la conexión y continuar con la operación. Esta interfaz incluye elementos visuales como capas semitransparentes y mensajes contextuales para mejorar la experiencia, además de mantener un bucle activo que actualiza en tiempo real el estado del control hasta que se detecta la acción requerida. En conjunto, la lógica principal del sistema se basa en la sincronización asíncrona mediante hilos para la detección USB, la persistencia y gestión del estado global para coordinar entre los componentes, y una robusta gestión de errores para garantizar la estabilidad durante operaciones críticas como el montaje, la copia y el desmontaje de dispositivos.

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

Esta sección del código se encarga de la emulación de juegos y la gestión de las pantallas previas a la ejecución. La función principal, *launch_game()*, inicia la emulación de un juego específico recibiendo la ruta del archivo ROM y el control conectado. Antes de lanzar el emulador, muestra una pantalla con el mapeo de controles correspondiente al sistema emulado, asegurándose de que el control esté conectado; si se desconecta, aborta el lanzamiento. Prepara la ejecución reiniciando la pantalla de *Pygame* y ocultando el cursor, luego lanza el proceso del emulador con *subprocess.Popen* y marca el estado global como emulador en ejecución. Después, llama a *monitor_emulator()*, que vigila el estado del proceso y la conexión del control durante toda la ejecución.

La función *monitor_emulator()* mantiene un ciclo activo mientras el emulador corre, verificando si el control permanece conectado. Si detecta desconexión, termina el proceso del emulador y actualiza el estado. Además, escucha eventos para detectar la combinación *SELECT* + *START*, que sirve para cerrar el emulador, terminando el proceso si se presionan simultáneamente. Este monitoreo se realiza con pausas cortas para no saturar el ciclo.

Por último, *show_mapping_control_screen()* muestra la imagen del mapeo de controles según la extensión del juego. Carga y muestra la imagen en pantalla completa, esperando que el usuario confirme la conexión presionando el botón A. Si el control se desconecta durante la espera, retorna False para indicar que no debe continuar la emulación. En caso de errores, también retorna False. En resumen, estas funciones aseguran que la emulación se inicie solo cuando el control esté conectado y configurado, y monitorean su estado durante la ejecución, proporcionando una experiencia robusta y controlada.

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

Hablando sobre el desarrollo e implementación de una interfaz grafica para asegurar la navegación clara y eficiente dentro de la retroconsola, se desarrollaron varios módulos los cuales en conjunto cumplen este cometido, tomando como baso los siguientes parámetros:

- **Pantalla de inicio (splash).** Hacer la implementación de una pantalla de bienvenida con un logo personalizado y sonido al iniciar la aplicación, generando una experiencia atractiva y profesional desde el primer momento.
- **Carga y visualización de caratulas.** Realizar una búsqueda de las caratulas de los juegos que se podrán emular, con el fin de mejorar la navegación visual y facilitar el reconocimiento de los títulos.
- **Organización y navegación de juegos.** Organizar los juegos por consola, permitiendo al usuario explorar los diferentes títulos disponibles dando a conocer a que consola pertenecen, tomando en cuenta que los controles de navegación se muestran en todo momento para saber cómo moverse entre los diferentes menús del programa.

6.3.1 Módulo de visualización

Para la parte de visualización del proyecto se desarrollaron funciones encargadas de crear la interfaz gráfica del mismo.

show_splash()

Esta función es la encargada de mostrar la imagen y sonido de inicio de la consola, esto se hace mediante el uso de pygame[3], donde primero inicializamos el módulo y creamos una ventana de 640x480 pixeles en modo pantalla completa (tomando en cuenta que esta será la resolución para toda la interfaz gráfica y la emulación de los juegos), de la siguiente manera:

```
[1] pygame.display.init()
[2] screen = pygame.display.set_mode((640, 480), pygame.FULLSCREEN)
```

Una vez se inicializo la ventana, se tiene que cargar la imagen que se va a mostrar, una vez cargada, se debe actualizar la interfaz de la siguiente manera:

```
[1] splash = pygame.image.load(/Ruta/de/la/imagen/que/se/mostrara)
[2] screen.blit(splash, (0, 0))
[3] pygame.display.update()
```

Así mismo, de una manera similar se implementa el sonido de inicio, inicializando el módulo de sonido y cargando el archivo que se va a reproducir:

```
[1] pygame.mixer.init()
[2] pygame.mixer.music.load(SPLASH_SOUND)
[3] pygame.mixer.music.play()
```

De esta forma se está inicializando la imagen y el audio de arranque del sistema, el llamado a esta función se realiza desde la función *main*, y es la primera en ejecutarse para asegurar que sea lo primero que se vea al iniciar la retroconsola, cabe mencionar que de una forma similar es como se muestran las imágenes de mapeo de controles descritas en los módulos de emulación.

[3] Pygame es una biblioteca para crear interfaces gráficas y juegos en Python.
Referencia. *pygame news*. (s. f.). https://www.pygame.org/news

draw_menu()

Esta función crea una interfaz gráfica cuidadosamente planificada para que la experiencia de usuario sea lo más amena posible, balanceando funcionalidad, rendimiento y estética, diseñada para ser controlada con D-Pad y optimizada para que funcione en pantalla completa tomando en cuenta el hardware limitado de la Raspberry Pi, esta función recibe 5 parámetros para poder dibujar el menú principal, los cuales son:

- **screen.** Superficie de Pygame sobre la cual se dibuja.
- **items.** Lista de elementos a mostrar (ROMs).
- **selected.** Indice del elemento seleccionado para mostrar su respectiva caratula.
- **current_path.** Ruta actual del repositorio para poder mandarla al momento de emular.
- **game_state.** Estado general de la aplicación, funciona como una bandera para saber si el menú se tiene que mostrar o no.

Para comenzar con el desarrollo de la interfaz se debe tener en cuenta la delimitación de espacios para colocar cada elemento, esto quiere decir que, se subdividirá el área total de la pantalla en 3 subáreas para mostrar el contenido e instrucciones, de la siguiente manera:

- **Área de título.** Se muestra un título personalizado (en este caso se muestra un letrero que dice “Todas las ROMS”)
- **Área de títulos disponibles.** Se muestran los títulos disponibles clasificados por consola, en esta área de la interfaz el usuario puede navegar de arriba abajo para recorrer la lista de títulos disponibles obtenida mediante el parámetro ítems.
- **Área de controles.** Se muestran los controles de navegación que pueden ser usados por el usuario.

Y para dibujar estas 3 áreas de interés, se hace uso (a grandes razgos) de la siguiente estructura de programación, donde se deben de declarar: la fuente del texto, el texto que se va a mostrar, el color del texto y las coordenadas de la ventana en la cuales se dibujara.

```
[1] font = pygame.font.Font(None, 24)
[2] text_surface = font.render("Texto a mostrar", True, (255,255,255))
[3] screen.blit(text_surface, (PosicionEnPantallaX, PosicionEnPantallaY))
[4] pygame.display.update()
```

De una forma similar se hace el dibujo de las caratulas de los juegos disponibles, en esta parte se toma en cuenta la dirección del elemento seleccionado actual, y se realiza la búsqueda de la imagen dentro de la carpeta donde estas se almacenan, la estructura utilizada es la siguiente, donde se deben declarar: la imagen que se mostrara dependiendo de una ruta, la posición en X e Y de la pantalla donde se dibujara, el largo y ancho de la caratula.

```
[1] cover_image = pygame.image.load(ruta_imagen)
[2] cover_area = pygame.Rect(460, 70, 165, 150)
[3] scaled_cover = pygame.transform.scale(cover_image, (nuevo_ancho, nuevo_alto))
[4] screen.blit(scaled_cover, (pos_x, pos_y))
```

draw_search_results()

Esta función crea una interfaz gráfica similar a la del menú principal, esta función recibe 2 parámetros para poder dibujar el menú principal, los cuales son:

- **screen.** Superficie de Pygame sobre la cual se dibuja.
- **game_state.** Estado general de la aplicación, funciona como una bandera para saber si el menú se tiene que mostrar o no.

Esta interfaz esta diseñada para mostrar solo las ROMs cuyo nombre concuerden con lo que el usuario busco dentro del menú de búsqueda, sigue la misma lógica de determinar áreas especificas para mostrar el título, lista de juegos a seleccionar y área de controles.

show_search_keyboard()

Esta función se encarga de crear una pantalla donde el usuario pueda realizar la búsqueda de algún juego en particular o con ciertos caracteres en el nombre del juego, crea un teclado virtual por el cual te puedes mover entre cada letra usando el D-Pad e implementa HotKeys que permiten realizar una búsqueda más ágil y eficaz, esta función recibe 2 parámetros para poder dibujar el menú principal, los cuales son:

- **screen.** Superficie de Pygame sobre la cual se dibuja.
- **game_state.** Estado general de la aplicación, funciona como una bandera para saber si el menú se tiene que mostrar o no.

En primera instancia se crea un teclado virtual que depende del parámetro game_state, ya que este parámetro es el que lleva el control global del estado en el cual se encuentra la ejecución del programa, esto debido a que, dependiendo de la acción realizada sobre el teclado, se desencadena otra, para comenzar, se debe de declarar la estructura de los caracteres que debe de contener este teclado.

```
[1] game_state.keyboard_layout = [
[2]     ['Q','W','E','R','T','Y','U','I','O','P'],
[3]     ['A','S','D','F','G','H','J','K','L'],
[4]     ['Z','X','C','V','B','N','M',"."],
[5]     ['SPACE','DEL']
[6] ]
```

Una vez declarada la estructura se tiene que hacer la implementación gráfica, tomando en cuenta un renderizado automático, el espacio de cada tecla contemplando las que van a tomar más espacio que otras y una correcta ubicación dentro del área de la pantalla para mayor formalidad, sin olvidar que, como en los menús descritos anteriormente, se debe de delimitar el área de cada parte de la pantalla (Titulo, teclado y controles), el renderizado del teclado se puede implementar de la siguiente manera:

```
[1] row_width = sum(
[2]     (key_width * (5 if key in ['SPACE'] else 3 if key in ['DEL'] else 1) +
[3]     key_margin * (4 if key in ['SPACE'] else 2 if key in ['DEL'] else 0))
[4]     for key in row
[5] ) - key_margin
```

Así mismo, se le puede aplicar un formato al teclado para que resalte la tecla seleccionada, colocar bordes redondeados y ajustar el borde de estos de la siguiente manera:

```
[1] is_selected = (row_idx, col_idx) == game_state.keyboard_selected
[2] key_color = COLOR_SEARCH_HIGHLIGHT if is_selected else COLOR_KEY
[3] pygame.draw.rect(screen, key_color, key_rect, border_radius=4)
```

6.3.2 Módulo de navegación

Para la parte de navegación del proyecto se desarrollaron funciones encargadas de interpretar la lógica de navegación, es decir, se encargan de interpretar los botones presionados durante la ejecución del programa y vincular estas pulsaciones a lo que gráficamente se tiene que mostrar, funciona como vínculo entre lo físico y lo gráfico, las funciones mencionadas son:

folder_menu()

Previo a la interpretación de los controles ya se debió de haber realizado el dibujo del menú principal para poder tener, en este caso, objetos que poder seleccionar, dentro de la función se implementa la siguiente línea para tener siempre el registro de la posición del elemento seleccionado:

```
[1] game_state.selection_history[game_state.current_path] = game_state.selected
```

Para implementar la navegación vertical y selección de algún título disponible, mientras se actualiza el valor del ítem seleccionado, dentro de la programación de esta función implementa lo siguiente:

```
[1] if hat[1] == 1:  # Flecha arriba
[2]     game_state.selected -= 1
[3] elif hat[1] == -1:  # Flecha abajo
[4]     game_state.selected += 1
```

De igual forma, dentro de esta función se implementa lo necesario para que al momento de detectar que se presiona el botón de busqueda (en el caso del control de Xbox Series S/X el botón ←), se ejecute la función encargada de mostrar el menú de búsqueda, tomando en cuenta que para el llamado de esta función es necesario actualizar las variables que se encargan de auditar el estado actual de la consola en todo momento, esto se realiza de la siguiente manera:

```
[1] elif hat[0] == -1:  # Izquierda (activar búsqueda)
[2]     game_state.search_active = True
[3]     game_state.search_text = ""
[4]     game_state.search_results = []
[5]     handle_search_menu(game_state.joystick, game_state)
```

Así mismo, cuando ya se eligió un juego y se detecta que el control presiono el botón de selección (en el caso del control de Xbox Series S/X el botón A), se tiene que ejecutar la función *launch*_game, enviando la ruta de donde el emulador va a tomar la ROM del juego y el identificador correspondiente al control de Xbox, de la siguiente manera:

```
[1] if event.button == 0:
[2]     item = items[game_state.selected]
[3]     launch_game(item[2], joystick)
[4]     screen = pygame.display.set_mode((640, 480), pygame.FULLSCREEN)
```

Finalmente, se implementa un sistema de apagado, el cual es activado al detectar la pulsación simultanea de dos botones, de la siguiente manera:

```
[1] elif event.button == 6 or event.button == 7:  # SELECT o START
[2]     if joystick.get_button(6) and joystick.get_button(7):
[3]             shutdown_raspberry()
```

handle_search_menu()

Esta función es la encargada de controlar el teclado virtual, permitiendo escribir términos de búsqueda mediante la navegación entre las teclas virtuales y tomando en cuenta las acciones al presionar botones en especifico que ayudan a facilitar la realización de la búsqueda, similar a la implementación dentro del menú principal, la posición del cursor (elemento/tecla seleccionado(a)) está en constante actualización, para que dependiendo del movimiento que haga el usuario dentro del teclado siempre se esté seleccionando la tecla correcta, esto se hace a grandes rasgos de la siguiente manera:

```
[1] hat = joystick.get_hat(0)  # (horizontal, vertical)
[2] row, col = game_state.keyboard_selected
[3] if hat[0] == 1: col += 1  # Derecha
[4] elif hat[0] == -1: col -= 1  # Izquierda
[5] elif hat[1] == 1: row -= 1  # Arriba
[6] elif hat[1] == -1: row += 1  # Abajo
[7] col = max(0, min(col, len(keyboard_layout[row])-1))
[8] row = max(0, min(row, len(keyboard_layout)-1))
[9] game_state.keyboard_selected = (row, col)
```

Así mismo, se implementa la lógica de selección de tecla:

```
[1] if event.button == 0:  # Botón A
[2]     row, col = game_state.keyboard_selected
[3]     key = game_state.keyboard_layout[row][col]
[4]     game_state.search_text += key.lower()
```

Al mismo tiempo, se consideran las acciones de los botones especiales para agregar caracteres de forma rápida y poder realizar una búsqueda más rápida y eficiente de la siguiente manera:

```
[1] elif event.button == 3:  # Botón Y (Espacio alternativo)
[2]     game_state.search_text += ' '
[3] elif event.button == 2:  # Botón X (Borrar alternativo)
[4]     game_state.search_text = game_state.search_text[:-1]
[5] elif event.button == 1:  # Botón B (Cancelar)
[6]     game_state.search_active = False
```

Finalmente, se implementa la lógica que se debe realiza en cuanto se presiona start (realizar búsqueda), donde se manda a ejecutar la función que muestra los resultados de búsqueda, tomando en cuenta lo escrito por el usuario, el estado actual del control y las variables que se encargan de auditar el estado actual de la consola de la siguiente manera:

```
[1] elif event.button == 7:  # Botón START (Buscar)
[2]     if game_state.search_text:
[3]       game_state.search_results = search_roms(game_state.search_text, ROM_DIR)
[4]       game_state.search_selected = 0
[5]       show_search_results_menu(joystick, game_state)
[6]       screen = pygame.display.set_mode((640, 480), pygame.FULLSCREEN)
```

show_search_results_menu()

Para finalizar con las funciones contempladas dentro del modulo de navegación, tenemos la función encargada de controlar lo que se realiza dentro del menú de los resultados de búsqueda, esta función tiene el mismo comportamiento que la función *folder_menu()*, con la diferencia de que aquí las ROMs que se muestra son solo las que coinciden con los resultados de búsqueda, es decir, los controles de navegación son los mismos, pero en este caso no se cuenta con el acceso directo para realizar la búsqueda de juegos, en su lugar se implementa la lógica para regresar al menú principal de la siguiente manera:

```
[1] elif event.button == 1:  # Botón B (Volver)
[2]     game_state.search_active = False
```

6.3.3 Módulo de gestión de contenido

Para la gestión del contenido de la retroconsola, se desarrollaron 3 funciones, las cuales ayudaran a leer el contenido existente para mostrarlo dentro de los menús, es decir, fungirán como las conexiones entre el contenido almacenado y el programa, buscando tanto las ROMs disponible como sus respectivas caratulas para que puedan ser mostradas dentro del menú principal y el menú de resultado de búsqueda descritos anteriormente, las funciones mencionadas son:

load_roms_and_folders()

Es la principal función encargada de la gestión del contenido, esta diseñada principalmente para obtener de manera correcta todas las ROMs disponibles con sus respectivas direcciones dentro del almacenamiento de la retroconsola.

Se cuenta con un sistema de mapeo por extensión, en otras palabras, un sistema que se encarga de clasificar automáticamente las ROMs por plataforma, esto principalmente para que al momento de mostrar los juegos disponibles dentro del menú principal se puedan mostrar ordenadamente y clasificados por consola, la clasificación se realiza de la siguiente forma:

```
[1] console_map = {
[2]     '.gba': 'GBA',
[3]     '.nes': 'NES',
[4]     '.smc': 'SNES',
[5] '.sfc': 'SNES' }
```

Y para crear la lista que contiene las ROMs junto con sus respectivas direcciones se sigue la siguiente lógica, tomando cuenta que lo que se busca es mostrar los juegos clasificándolos por consola:

```
[1] roms = []
[2] for root, _, files in os.walk(ruta_roms):
[3]   for file in files:
[4]   ext = os.path.splitext(file)[1].lower()
[5]   if ext in {'.gba', '.nes', '.smc', '.sfc'}:  # Filtra extensiones
[6]    consola = {'gba': 'GBA', 'nes': 'NES', 'smc': 'SNES', 'sfc': 'SNES'}[ext[1:]]
[7]    roms.append((consola, file, os.path.join(root, file)))
[8] roms_ordenadas = sorted(roms, key=lambda x: (x[0], x[1]))
```

search_roms()

Esta función se encarga de buscar ROMs en el sistema de archivos filtrado por nombre y extensión, organizando los resultados por consola para poder ser mostrados en el menú de búsqueda, en primer instancia filtra por nombre y consola las ROMs disponibles en el sistema al momento de que el usuario realiza la búsqueda de un juego, posteriormente clasifica los resultados basándose en las extensiones de estos para finalmente devolver una lista de tuplas con las ROMs clasificadas y ordenadas de las siguiente manera:

```
[1] for root, _, files in os.walk(root_dir):          # Búsqueda recursiva
[2]     for file in files:                                # Examinar cada archivo
[3]         file_lower = file.lower()
[4]         for ext, console in console_map.items():  # Verificar extensión
[5]             if file_lower.endswith(ext) and search_lower in file_lower:
[6]                 results.append(('rom', file, os.path.join(root, file), console))
[7]                 break  # Evita duplicados si hay múltiples extensiones
```

load_game_cover()

La función busca y carga la imagen de portada de un juego, esta imagen es mostrada tanto en el menú principal como en el menú de resultados de busque y tiene la finalidad de permitir al usuario reconocer el titulo del juego seleccionado, esa búsqueda se realiza dependiendo del nombre del juego y la extensión del mismo, ya que las caratulas de los juegos se encuentran en directorios particulares por consola, la lógica que se sigue para realizar la búsqueda es la siguiente:

```
[1] for ext, console in console_map.items():  # 1. Clasificar por consola
[2]   if game_name.lower().endswith(ext):
[3]     cover_dir = f"/.../covers/{console}"  # Ruta dinámica
[4]     for filename in os.listdir(cover_dir):  # 2. Buscar coincidencias
[5]       if base_name in filename.lower():  # 3. Comparación flexible
[6]           if any(filename.lower().endswith(img_ext) for img_ext in extensions):
[7]             return pygame.image.load(os.path.join(cover_dir, filename))
```

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

El desarrollo de la retroconsola **ccjpmmGaming** en Raspberry Pi 4 con Raspberry Pi OS Lite demostró cómo sistemas embebidos pueden integrar hardware accesible, software eficiente y diseño centrado en usuario para revivir la experiencia de videojuegos clásicos. Combinando Python con bibliotecas como *Pygame* y *Pyudev*, se implementó un sistema automatizado que detecta y copia ROMs desde USB, organizándolas por consola (GBA/NES/SNES), mientras una interfaz gráfica optimizada permite navegación fluida con gamepad, incluyendo menús dinámicos con carátulas y teclado virtual para búsquedas. El uso de *Mednafen* como emulador principal, gestionado mediante subprocesos, aseguró equilibrio entre rendimiento y consumo de recursos, superando limitaciones típicas de hardware embebido. Más allá de su función lúdica, el proyecto destaca como caso de estudio en programación de sistemas embebidos, mostrando técnicas para manejo de periféricos, concurrencia y optimización de memoria. Su código modular y documentación detallada en *GitHub* lo convierten en base para expandir funcionalidades (como soporte para más consolas) o adaptarlo a fines educativos, evidenciando que la preservación digital de videojuegos retro puede lograrse con soluciones técnicas elegantes y hardware económico. Los retos enfrentados - desde cuellos de botella en emulación SNES hasta compatibilidad de controles - subrayan la importancia del diseño iterativo, mientras futuras mejoras potenciales (integración con EmulationStation o streaming) prometen ampliar su alcance. En esencia, ccjpmmGaming trasciende como proyecto técnico para convertirse en puente entre nostalgia y tecnología moderna, demostrando que la innovación en sistemas embebidos puede democratizar el acceso al patrimonio digital de los videojuegos.

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*.