# RetroPlay: Consola retro de entretenimiento

Hernández Jaimes Rogelio Yael

Fundamentos de Sistemas Embebidos, Grupo: 03

Repositorio GitHub: https://github.com/roshercs/RetroPlay.git

Vídeo demostrativo de ejecución: https://youtu.be/u4oxUGFrXT0

05 de Diciembre, 2023

## 1. Objetivo

El lector aprenderá a configurar la tarjeta Raspberry Pi para ejecutar un centro de entretenimiento capaz de emular la ejecución de algunos tipos de consola definidos.

## 2. Introducción

Se implementará un centro de entretenimiento que permita la emulación de las consolas NES, SNES y GameBoy Advance. La emulación directa se realizará haciendo uso de Mednafen. El sistema será capaz de manejar las peticiones del usuario para selección de juego y su debida instancia a Mednafen. El menú de selección mostrará todos los juegos previamente cargados a la memoria del sistema. Así mismo, se tendrá un detector de USB que al detectar una conexión busca los archivos roms con extensiones '.nes','.snes' y '.gba', compara con los ya cargados. De no detectarse repetición de rom, añade el juego a la biblioteca creando una copia en memoria. El control del sistema se hará mediante el uso de un controlador propio (detallado posteriormente). El usuario podrá seleccionar los juegos y emular mediante Mednafen con dicho gamepad.

## 3. Antecedentes

Las consolas retro (ej. NES, SNES, GameBoy Advance, entre otros) son dispositivos de entretenimiento muy populares en su momento. Sin embargo, por el mismo paso de tiempo su disponibilidad se ha visto muy reducida. Así pues, la funcionalidad de estas consolas puede ser reproducida por software por los equipos de cómputo actuales. Mednafen es un emulador multiplataforma instanciado desde linea de comandos, que utiliza OpenGL y SDL, capaz de emular un total de 17 consolas, entre las que se encuentra la NES, SNES, GameBoy Advance, Atari Lynx, entre otros[1].

Para la realización de este proyecto se planteó además un menú interactivo a través del cual el usuario puede hacer la selección del juego a ejecutar. Para este fin se empleó la biblioteca tkinter de Python la cual provee un kit de herramientas para la generación y manejo de interfaces de usuario gráficas (GUI). La biblioteca tkinter están disponibles en la mayoría de las plataformas Unix, así como en sistemas Windows [4].

El sistema está diseñado para operar con la tarjeta Raspberry Pi 4 (o Pi 3). La tarjeta Raspberry Pi 4 es una computadora de escritorio desarrollada por la Raspberry Pi Foundation. Esta computadora se encuentra disponible en distintas versiones, variando principalmente elementos como la cantidad de RAM (1Gb, 2Gb, 4Gb u 8Gb). La Raspberry Pi 4 integra en una misma placa integra puertos de mucha utilidad como lo son: puerto micro HDMI, puertos USB 2 y USB 3, y un puerto GigabitEthernet. [6] Para el funcionamiento de esta computadora, se requiere el uso de un sistema operativo, La fundación Raspberry reconoce la existencia de una gran variedad de sistemas operativos disponibles compatibles en la tarjeta integrada. Sin embargo, se sugiere el uso del sistema operativo oficial: Raspberry Pi OS. Este sistema es el que se empleará para este proyecto. El conector GPIO de la raspberry resulta clave para el manejo del sistema puesto que es por estos pines que se conectaría el control gamepad.

## 4. Materiales

Para el desarrollo de este proyecto, y como se hizo mención previamente, se requiere el uso de la tarjeta Raspberry Pi 4, Raspberry Pi 3, o incluso podría evaluarse el uso de la Raspberry Pi 5 (introducida al mercado poco tiempo a la fecha). Además, se requiere el uso del sistema Raspberry Pi OS con Desktop. Este es necesario para la creación y manejo de las interfaces gráficas. Adicionalmente, se plantea un circuito sencillo que fungirá como gamepad para el control del sistema. Para este, en adición a los requeridos para el uso de la raspberry, se requieren los siguientes materiales:

- Adaptador cargador USB (10–15W) con salida 5V@3A (sin carga rápida)
- Cable USB-C con soporte para transporte de datos

- Adaptador mini-HDMI a HDMI o Adaptador HDMI a VGA o DVI3 (Para Raspberry Pi 4 y Raspberry Pi 5, y dependiendo de la entrada de la pantalla a utilizar)
- Jumpers de conexión Macho-Hembra (Aproximado 15): a mayor tamaño más rango de uso del gamepad.De necesitarse mayor longitud, podría optarse por el uso de cable de circuito (distinto color recomendado) junto con conectores para cable dupont macho y hembra.
- 13 resistencias de 330Ω
- 13 botones tipo push-botton de dos patas. En caso de adquirir botones de 4 patas, se indica el uso de dos patas opuestas, por ejemplo, dos pines en diagonal.
- Protoboard
- Memoria Micro-SD de 16 GB o más.

Se asume que el lector tiene conocimientos básicos sobre el alambrado de circuitos, mas no especializados.

## 5. Funcionamiento de los componentes electrónicos relevantes

No se hace uso de circuitos integrados u otros componentes más complejos para la sencillez de implementación para el usuario. El circuito implementa circuitos de botones simples, donde se conectan los pines de un botón a 5V y a tierra, donde al presionar un botón se permite el paso de corriente. El esquema básico de este circuito es el siguiente: Entiéndase que estos circuitos son circuitos simples pull-up. Los puntos de referencia, A y B son de hecho a donde

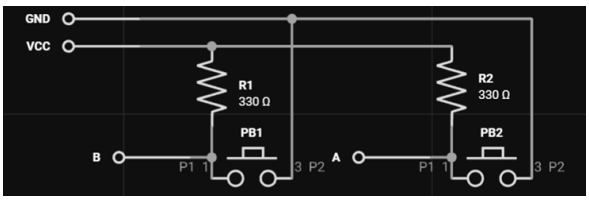


Figura 1: *Circuito simple de botones pull-up*

habrían de conectarse los pines GPIO para operar en el programa. La resistencias conectadas entre los pines y la fuente de alimentación (5V salida de la Raspberry) permitirá mantener los puntos de referencia en estado alto mientras el botón no esté pulsado. Al pulsarlo los puntos de referencia pasan a estado bajo en consecuencia. Estos circuitos permiten evaluar de forma más precisa ambos estados necesario para la implementación de este proyecto.

Cada botón en el gamepad será un circuito simple como el mostrado. Si bien, los puntos de referencia son donde habrían que conectarse los pines de la Raspberry, se recomienda conglomerar estos puntos a una misma zona para facilitar la conexión a tarjeta así como evitar enredos entre los cableados. En adición a esto, los botones pueden ser mapeados a nuestra conveniencia, así como también pueden colocarse en protoboard distribuidos en cualquier forma. Sin embargo, se recomienda la siguiente distribución:

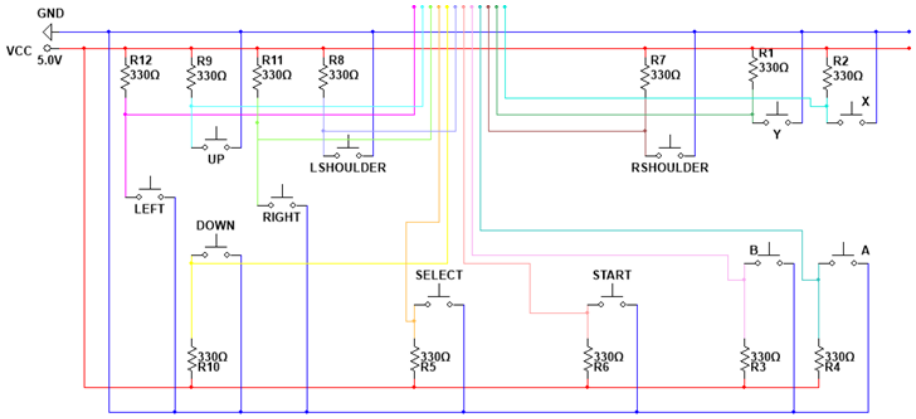


Figura 2: *Distribución sugerida de botones*

Nótese que todos los puntos de referencia se conglomeran en una misma zona, esto hará que los cables conexión a la Raspberry estén juntos y puedan manipularse más fácilmente. La asignación de los pines GPIO de la tarjeta a la protoboard se plantearon siguiendo el orden del circuito base. De esta forma, el primer pin de libre de la raspberry (que no sea 5V, GND o 3,3V) se asigna a la tecla Left. La asignación de los pines a cada botón es la siguiente:

```
#Variables Controlador
boton_L=3
boton_U=5
boton_R=7
boton_Sel=8
boton_D=10
boton_LS=11
boton_Start=12
boton_B=13
boton_A=15
boton_RS=16
boton_Y=18
```

boton_X=19

boton_Ex=22

Esta numeración sigue el modo BOARD definido en la documentación de la Raspberry. De esta forma, el GPIO 15 corresponde a la acción del botón A. Mientras se respete el mapeado de este punto, el gamepad funcionará de forma correcta.

# 6. Funcionamiento de la tarjeta controladora (Raspberry Pi

El uso de la Raspberry Pi 4 es sencillo, bastante parecido al uso de una computadora como se conocen en la actualidad. El adaptador cargador requiere operar con 10-15W. La salida del mismo deben ser los 5V con los 3A e indispensablemente que no tenga carga rápida. De no emplear un cargador con estas características se arriesga a quemar la tarjeta. Si la corriente de salida del cargador es 1A podría funcionar solamente si se emplea el sistema operativo Raspberry Pi OS Lite (operada a linea de comando) con posibilidad de no tener energía suficiente para operar. Con 2A puede operar pero tendría carencia de energía que sería notificado y podría hacer que la tarjeta se apagara al momento de ejecutar instrucciones más complejas. Los 3A permiten a la tarjeta operar plenamente y son requeridos para este proyecto dado el consumo de recursos que implican los procesos gráficos. La tarjeta opera con un sistema operativo conectado por el puerto de micro-sd en la parte interior de la tarjeta. Este método de uso genera un menor recurso de potencia y mayor velocidad de lectura en la Raspberry, lo que nos conviene para evitar retrasos en las ejecuciones.

La Raspberry Pi 4 cuenta con distintos puertos disponibles, como 2 puertos USB 2.0 y 2 USB 3.0, así como dos puertos micro-HDMI, puerto Ethernet y los conectores GPIO. Para operar la Raspberry se requiere conectar un teclado y mouse como si fuese cualquier computadora. El conector GPIO se compone de un conjunto de pines. En el caso particular de la Raspberry Pi 4 se tiene un total de 40 pines. Cualquiera de los pines GPIO puede ser configurado como entrada o salida (exceptuando pines GND, 5V y 3.3V). Además de esto, algunos pines poseen funciones específicas. Por ejemplo: función PWM (pin 12 y 13), puerto serial (TXD: pin 14, RXD: pin 15), entre otros [5]. Para el uso de los pines GPIO en Python tenemos distintas bibliotecas disponibles. Entre estas encontramos RPi.GPIO. Este paquete provee a Python control sobre los pines GPIO en una Raspberry Pi. Sin embargo, de acuerdo con su proveedor, no soporta funciones SPI, I2C, ni comunicación serial[3]. Para su uso, la instrucción *GPIO.setmode(GPIO.BOARD)* indica que el conector GPIO tendrá la asignación de pines indicado por la propia documentación de la tarjeta Raspberry. Por su parte, la instrucción *GPIO.setup(channel, state)*, indica que el pin *channel* opera en modo *state* (GPI.IN: entrada y GPIO.OUT: salida). Otra función utilizada es *GPIO.output(n, state)* que define el estado de salida del pin *n*, bajo (0, false, GPIO.LOW) o alto (1, True, GPIO.HIGH). El programa se encarga de configurar estos pines para que se

manejen como entradas, además de que se definen con el número de pin que se verá más adelante.

## 7. Información para la salud y cuidado de componentes

El uso de la consola podría conllevar la reproducción de juegos con destellos constantes. Si sufre de ataques epilépticos, consulte con un médico antes de hacer uso del sistema. No existe riesgo de quemaduras ni electrocución severa al hacer uso del sistema ni el circuito del gamepad asociado. Sin embargo, podría experimentar una electrocución leve en caso de recargar las manos sobre las resistencias (conectadas a corriente) o si se tocaran los pines de 5V y tierra al mismo tiempo.

Tanto como para evitar posibles daños a la salud como el cuidado de los componentes. Se recomienda el uso de materiales aislantes (ej. cinta de aislar) para recubrir las conexiones en protoboard una vez comprobado el correcto funcionamiento del gamepad. Con este recubrimiento se evita el contacto directo con los componentes y conexiones además que evitan que los componentes sufran desconexiones accidentales por la manipulación del gamepad.

Otras precauciones para el cuidado del sistema recae en el uso de los pines y jumper en estos. Se recomienda el uso de la tarjeta raspberry sobre una superficie plana y firme. Así mismo, en función de la longitud del cableado para la conexión del gamepad a la raspberry se recomienda mantenerse a una distancia lo suficiente para mantener una tensión holgada en estos, para evitar así la presión en los pines GPIO y jumpers conectados a la protoboard.

Así pues, para evitar daños en la Raspberry, se hace énfasis en el manejo con cuidado de la salida de 5V y tierra. el cruce de ambos puede generar un corto que desencadena en el apagado forzado de la Raspberry. Para evitar este tipo de situaciones se recomienda conectar cada pin (GND y 5V) uno a la vez a la protoboard, evitando el menor contacto entre sí.

## 8. Configuración de la tarjeta controladora

Para poder utilizar la tarjeta se requiere instalar el sistema operativo Raspberry Pi OS en su versión con Desktop para 64 bits. La instalación inicial del mismo solo requiere seguir los pasos indicados por el propio proceso de configuración. Para la configuración de la tarjeta, se sugiere la descarga del repositorio de github.Se asume que el usuario realiza la instalación base del sistema operativo Raspberry Pi OS con escritorio de 64 bits.

## 8.1. Instalación automática

Para la instalación del proyecto de forma más automatizada, requerimos crear la carpeta gitCopy en la ruta /home/pi. Para ello empleamos el comando sudo mkdir /home/pi/gitCopy. Una vez creada la carpeta descargamos el directorio de github (se asume que la tarjeta ya cuenta con conexión a internet). Para descargar el repositorio se requiere instalar los comandos de git, por lo que primero requerimos actualizar apt-get con el comando sudo apt-get update. Con este actualizado ejecutamos el comando

```
apt-get install git
```

Para habilitar los comandos de gitHub. Con los paquetes instalados se clona el repertorio en la carpeta creada con el comando:

```
git clone https://github.com/roshercs/RetroPlay.git /home/pi/gitCopy
```

Esto copia el contenido del repositorio dentro de nuestro sistema. Con los archivos descargados procedemos a ejecutar la instalación automática. Nos debemos posicionar en la carpeta gitCopy (cd /home/pi/gitCopy). Dentro de este escritorio habilitamos el modo super usuario con el comando sudo su. Se conceden los permisos permitentes a la instalación con el comando:

```
chmod +x install.sh
```

Con los permisos concedidos, ejecutamos el comando para ejecutar:

```
./install.sh
```

## 8.2. Instalación manual

Los primeros cambios requeridos es la actualización de las herramientas de instalación de paquetes. En otras palabras: apt-get -update y apt-get upgrade. El primero actualizará el instalador de paquetes mientras que el segundo actualizará los paquetes instalados. Esto se hace para poder obtener adecuadamente todos los paquetes necesarios para el proyecto. Dada las políticas de seguridad de Rasbperry Pi OS, no es posible instalar bibliotecas de python mediante la herramienta pip. En muchos casos basta con ejecutar el equivalente comando apt install python-nombrePaquete. Sin embargo, existen paquetes que no están disponibles por este medio. Es por esta razón que se crea un entorno virtual con python. Primeramente, creamos las carpetas roms", ïmages", RetroPlay.[en] la ruta base /home/pi/". Sobre estas trabajará el sistema para emular los archivos ROM, el manejo de imágenes y del sistema en general. usamos el comando mkdir para crear estas carpetas.

Así pues, instalamos la primer paquete que permite la creación de los entornos virutal: sudo apt-get install python3-venv. Nos posicionamos en la carpeta de RetroPlay con cd /home/pi/RetroPlay y se crea un entorno virtual en python con el comando python3 -m venv RetroPlay. Para activar el entorno virtual se usa el comando `source /home/pi/RetroPlay/bin/activate`. Con este se accede al entorno virtual, por lo que en este punto ya se pueden instalar paqueterías con la herramienta pip install.

Primeramente, actualizamos tkinter con el pip install. Esta biblioteca se emplea para la creación de las interfaces gráficas como el menú de selección. Además, se instala o actualiza según sea el caso la biblioteca pillow (importada como PIL en el proyecto) para los módulos Image e ImageTk para el manejo de imágenes como las portadas de consolas de cada juego en el menú de selección. Otra biblioteca indispensable del proyecto es la biblioteca RPi.GPIO, para el control de los pines GPIO por donde controlaremos el sistema. El comando para esta es realizado mediante sudo apt-get install python-rpi.gpio.

Otra biblioteca a instalar es keyboard (pip install keyboard), este nos permite emular un teclado mediante funciones como press() y release(). Mediante estas es que simulamos la pulsaciones de teclas al momento de presionar los botones conectadas a los pines GPIO. Por su parte, pyudev (pip install pyudev) nos permite monitorear la conexión de dispositivos USB para la copia de archivos ROM. La última biblioteca a instalar es vlc (pip install python-vlc) para la reproducción de imágenes y vídeos, en este caso para la reproducción del logo inicial. El detalle de implementación de cada biblioteca se verá posteriormente en el desarrollo de componentes de software.

Dejando de lado las bibliotecas propias a usar en python, otros elementos a instalar es Mednafen. Este puede instalarse fuera del entorno virtual mediante sudo apt-get install mednafen. Este emulador es inicializado desde linea de comandos con la estructura mednafen ruta-rom. Para este requerimos hacer unos cambios. Para estos abrimos el archivo de configuración con nano `~/.mednafen/mednafen.cfg`. Dentro de este, buscamos la opción sound.device y reemplazamos default por hw:2. Este cambio genera que mednafen tome la salida del jack 3.5mm como salida de audio. Esto es para evitar el conflicto de audio existente en Linux como lo dice su propia documentación [2]. Además, se modifica video.fs a 1 para forzar la pantalla completa siempre. Guardamos estos cambios y ya se podría emplear mednafen adecuadamente.

Por otra parte, como se verá mas adelante, se requiere el uso de un teclado virtual, el cual generamos con la biblioteca matchbox-keyboard (apt install matchbox-keyboard). Este permite emular el teclado virtual sin la necesidad de tener un teclado físico conectado, lo que nos permitirá comunicarnos con mednafen a través de nuestro gamepad.

Ahora bien, para que al inicializarse se ejecute directamente al iniciar la raspberry necesitamos crear unos

directorios extras. Dentro de la carpeta /home/pi/.config creamos la carpeta autostart. En esta se crea el archivo inicio.desktop (necesario deifnir esta terminación como se indica). En el archivo creado ingresamos como primera línea [Desktop Entry] (respetando mayúsculas) y a partir de la segunda linea los comandos a ejecutar usando el parámetro Exec=. En nuestro caso, ya que requerimos inicializar el entorno virtual y ejecutar el archivo, tendríamos:

```
[Desktop Entry]
Exec=sudo su
Exec=source /home/pi/RetroPlay/bin/activate
Exec=cd RetroPlay
Exec=python3  RetroPlay.py
```

Con esto, nos aseguraríamos que el emulador se inicialice directamente apenas enciende la raspberry.

# 9. Desarrollo de componentes de software (módulos)

## 9.1. Reproducción de animación o imagen estática

Este modulo hace uso de la biblioteca vlc para la reproducción de la animación inicial del logo. Para ello se inicializa vlc, se instancia un reproductor, se carga el vídeo (`/home/pi/RetroPlayLogo.mp4`) y se carga al reproductor. Finalmente se reproduce y pasados 4 segundos (pasada la duración del vídeo) se cierra el reproductor. Esto se engloba en una sola función:

```
def reproduce_logo():
    ruta_logo= '/home/pi/RetroPlayLogo.mp4'
    sleep(1)
    instancia = vlc.Instance('--no-xlib')# Inicializar VLC
    reproductor = instancia.media_player_new() # Crear reproductor
    # Cargar el video
    medio = instancia.media_new(ruta_logo)
    reproductor.set_media(medio)
    # Reproducir el video
    reproductor.play()
    sleep(4)
    reproductor.stop() #Cierre de reproductor
```

## 9.2. Arranque directo a emulador

Para este fin se creo el archivo inicio.desktop detallado en la sección de configuración de tarjeta. Este archivo ejecuta la secuencia de comandos dados al inicio de la tarjeta, desplegando así el menú de selección, reproducción del logo y demás funcionalidades del sistema.

Una nota tomar en cuenta es que en el resto de módulos que se detallan fue necesario plantear los comandos a ejecutar haciendo uso de rutas absolutas a los distintos componentes, para de esta forma asegurar su correcta ejecución. Por ejemplo, en Mednafen se empleó la ruta `/usr/games/mednafen` para acceder a este de forma asegurada. Además a todos estos comandos se añadió la sentencia sudo para asegurar su ejecución sin conflictos con el sistema.

## 9.3. Control Completo de la consola usando gamepad

El control completo se realiza mediante el gamepad definido con el circuito base así como la asignación de los botones a los pines de la raspberry. A nivel de software el gamepad creado emulará un teclado en el sistema, de modo que al pulsar un botón el sistema lo reconocerá como la pulsación de una tecla del teclado. Esto es así para mantener una homogeneización entre el uso del menú de selección y Mednafen. Si bien en el menú podría controlarse detectando cambios en el estado de los pines de forma directa, Mednafen no reconoce estos cambios como posibles entradas. Al emular un teclado el sistema no solo reconoce las entradas a los pines para el menú de selección sino que permite operar mednafen como si del teclado se tratase. Para esto se emplea la biblioteca keyboard, la cual emula el comportamiento de los teclados disponibles en sistemas (tanto físicos como virtuales). Se plantea el uso de dos hilos: control de acción y control de movimiento. El primero se encargará de los botones de acción como A, B, X, Y, Left shoulder y Right Shoulder. Mientras que el control de movimiento se encarga del control de los botones Up, Down, Left, Rigth, Start, Select y Exit. Si bien, estos tres últimos no son necesariamente ligados al movimiento, se plantean aquí para una mejor distribución de carga. Ahora bien, se plantea el uso de dos hilos por la naturaleza del gamepad. El usuario tendrá mayor probabilidad de usar la tecla Left combinado con A que combinar dos teclas de acción. Por ejemplo en Mario Bros el usuario puede saltar mientras se desplaza a la izquierda, pero si trata de pulsar A y B al mismo tiempo se ejecuta una acción y luego la otra. Esto requirió precisamente de dos hilos que evalúen distinta situación.

La detección del botón pulsado utiliza el cambio de estado pull-up planteado en el circuito. Mientras no se pulse el botón el estado será leído en algo, mientras que si se pulsa el estado será bajo. Con esto cada hilo pregunta el estado de cada botón asociado y en caso de encontrarlo en estado bajo se emula que se ha pulsado una tecla. Este comportamineto puede verse en el siguiente ejemplo:

```
#Gestion Boton A
estado_actual=GPIO.input(boton_A) #Se lee el estado actual del boton asociado a A
```

```
if estado_actual==GPIO.LOW: #Si se encuentra en bajo significa que se pulso el boton
    keyboard.press(30)  #emula la pulsacion de la tecla A de teclado
    sleep(0.2)
    keyboard.release(30)    #se libera la pulsacion
```

Al igual que con el botón A se evalúa de uno en uno el estado de los demás botones. La función press y release puede operar con parámetros el nombre de tecla o su número. Se opta por este último pues da mejores resultados al momento del inicio automático del programa. La asignación de las teclas a cada botón responde directamente al nombre del botón. Por ejemplo, el botón A se asigna la tecla A, mientras que al botón Up se asigna la tecla Up (flecha superior). Las excepciones son la tecla Exit (mapeado a Escape), la tecla select(mapeado a space) y start (mapeado a Enter). Con estos mapeos, se pueden asignar estas a funciones de menú como el cambio de juego seleccionado y otros.

Ahora bien, dado a que la consola no puede tener teclado físico conectado, se hace uso de un teclado virtual generado por matchbox-keyboard, el cual requiere un tercer hilo para ejecutarse en paralelo. La función encargada de inicializar el teclado es la siguiente:

```
def teclado_virtual():
    comando='sudo /usr/bin/matchbox-keyboard' #ejecucion de teclado virtual
    proceso=sp.run(comando,shell=True)
    try:
        proceso.wait()
    except:
        print("Fin teclado virtual")
```

Como se mencionó en el arranque automático, este tipo de comandos requieren el uso de sudo y la ruta absoluta de la biblioteca para operar correctamente. El hilo inicializa el teclado y se queda en espera a que finalice.

## 9.4. Set de roms precargadas

Para este fin se tienen funciones de lecturas de archivos y filtrado de extensiones. Particularmente para la carga inicial de las roms almacenadas tenemos la función load_star_roms. Esta función lista todos los archivos existentes en la carpeta de roms (`/home/pi/roms`) y evalua el tipo de cada una. Primero se añaden aquellos juegos de NES, posteriormente los juegos de SNES y por último los juegos de GameBoy Advanced. Estos juegos se añaden a un arreglo que lista los juegos disponibles. La función general de esto es la siguiente:

```
    # Lista todos los archivos en la carpeta
    archivos_en_carpeta = os.listdir(ruta_roms)
    #Lectura de roms NES
    archivos_nes = [archivo for archivo in archivos_en_carpeta if archivo.endswith('.nes')
    for rom in archivos_nes:
        lista_juegos.append(rom)
```

Como con los archivos de extensión '.nes' se añaden los juegos '.sfc' y '.gba'. La lista_juegos es empleada en la generación del menú. Este menú se genera usando tkinter. Se crea una instancia del mismo, se configuran sus atributos para iniciarse maximizado y con otras características especiales. Así mismo se bindean algunas teclas (ej. Up) a funciones específicas que permiten el manejo del menú con teclas (mapeadas en la subsección anterior). La función actualiza_lista crea un frame donde ingresa una parte de los juegos listados de acuerdo al número de items por página. La idea es de acuerdo al juego seleccionado actualmente se imprime un bloque de los juegos u otro. Por ejemplo, con la lista de 15 roms precargadas y 4 items por página, al momento de pasar al item 5 se eliminarían los elementos actuales del frame y se reimprimen los juegos del 5 a 8. Esto se ve en la función:

```
def actualizar_lista():
    global frame
    # Limpiar la lista actual
    for widget in frame.winfo_children():
        widget.destroy()
    # Determinar la pagina actual
    page = selected_index // items_per_page
    start_index = page * items_per_page
    # Mostrar los elementos siguientes en el arreglo
    for i in range(start_index, min(start_index + items_per_page, len(lista_juegos))):
        juego = lista_juegos[i]
        [...]
```

Los cambios en el valor de selected_index se realizan mediante las teclas UP y DOWN (ligadas en la ventana de tkinter), cuando se pulsa una u otra se decrementa o incrementa el valor respectivamente. Tras la modificación se evalúa si el nuevo valor sale de los limites de la pagina (star_index+-selected_index) y de hacerlo se hace un cambio de página que retoma lo descrito anteriormente.

## 9.5. Detección de USB y carga de nuevas ROMS

Para esto primeramente establecemos una función que se encuentre a la detección constante de la conexión de una USB.La función detection_usb emplea la biblioteca pyudev para este fin. Una vez detectando la conexión de una USB se obtiene el punto donde se montó y se evalúa la posibilidad de subir los nuevos juegos. Para esto, se listan todos los archivos en la USB y se extraen aquellos de extensión '.nes','.sfc' y '.gba'. Cada uno de estos archivos extraídos es comparado con las rom ya cargadas. Esta comparación se realiza mediante el hash del nuevo archivo contra el hash del archivos precargados. En caso de detectar un hash repetido, se determina que el juego ya existe dentro de las rom cargadas (aun si tiene nombre diferente de archivo) y no se añade. En caso de no coincidir en el hash, se determina que no existe un juego igual por lo que haciendo uso de la biblioteca shutil copiamos la rom evaluada a la carpeta roms de la raspberry. Esto puede verse a groso modo en las siguientes funciones:

```
def detection_usb():
    #Instancias necesarias de pyudev
    context = pyudev.Context()
    monitor = pyudev.Monitor.from_netlink(context)
    #Se detectan aquellos que sean nuevas particiones (USB)
    monitor.filter_by(subsystem="block")
    #Se registran las actividades del monitor dado
    for device in iter(monitor.poll,None):
        #Si se detecta una conexion (add) se procede a operar
        if device.action=='add' and 'ID_FS_TYPE' in device:
          if 'usb' in device.get('ID_BUS'):
            [...]
            usb_path=get_mount_point(nomDispo)+'/' #Obtenemos punto de montaje
            upload_games() #Evaluan nuevos juegos
            [...]
                print_list_games()
```

Por su parte, uload_games usa una arreglo de extensiones (con las definidas previamente) y revisa la extensión de cada archivo listado, si coincide con alguna extensión envía a revisión.

```
def upload_games():
    extensiones=['.nes','.gba','.sfc']
    # Lista todos los archivos en la carpeta
```

```
    archivos_en_carpeta = os.listdir(usb_path)
    for archivo in archivos_en_carpeta:
        if archivo.endswith(tuple(extensiones)):    #extrae aquellos con las extensiones dadas
            compare_roms(archivo)
```

La función compare_roms obtiene el hash de cada juego ya cargado y lo compara con el hash del posible nuevo juego, si coincide con por lo menos uno, se omite su carga, si no coincidiera se añade a la lista y se copia a la carpeta roms:

```
def compare_roms(archivo_1):
    hash_archivo_1 = calcular_hash(usb_path+archivo_1)
    for archivo_2 in lista_juegos:
        hash_archivo_2 = calcular_hash('/home/pi/roms/'+archivo_2)
        if hash_archivo_1 == hash_archivo_2: #se repite hash?
            print("Juego repetido")
            return #Si, omite juego
    juegoDetectado=1 #bandera de juegos nuevos
    #Si no se coincide ningun hash se agrega el nuevo juego
    print(f"Se ha encontrado un nuevo juego: {archivo_1}")
    lista_juegos.append(archivo_1)
    shutil.copy(usb_path+archivo_1, ruta_roms) #copia a carpeta roms
    nuevos_juegos.append(archivo_1)
```

Si se activa por la bandera de nuevo juego entonces se requiere imprimir la lista de todos los nuevos juegos añadidos. Si mednafen no se encuentra ene ejecución la impresión es inmediata, pero si mednafen está ejecutándose se retiene a fin de mantener la integridad de la pantalla completa de mednafen. Una vez concluyendo la ejecución de esta se imprime en pantalla el listado.

## 10. Integración en modulo único

Dado a que el desarrollo de cada submódulo se hizo de forma independiente, la integración en un solo modulo requirió poco más que copiar las funciones en un mismo archivo y englobar algunas variables globales. Por ejemplo, en la detección de usb se añadió la bandera de runMednafen para evitar la impresión de lista al estar ejecutando Mednafen. Así mismo, existen otras variables que desde el diseño de cada módulo se planteo de carácter global, como el usb_path, necesario en distintas funciones para acceder a los archivos de la memoria. Dado a que el menú opera con las roms

leídas por el módulo de carga de usb, es desde la función menú que se hace la carga inicial de las roms.

Por otra parte para la ejecución de los distintos módulos se requirió el uso de distintos hilos. La forma general es la siguiente:

```
def compare_roms(archivo_1):
    try:
        repImg = threading.Thread(target=reproduce_logo)
        repImg.start()
        wait_event.set()
        teclado = threading.Thread(target=teclado_virtual)
        teclado.start()
        hilo1 = threading.Thread(target=manejo_control_movimiento)
        hilo1.start()
        hilo2 = threading.Thread(target=manejo_control_accion)
        hilo2.start()
        detectorUSB = threading.Thread(target=detection_usb)
        detectorUSB.start()
        menu()
    except:
        pass
```

Cada hilo plantea para un fin distinto. La reproducción del logo es única por lo que una vez concluyendo se libera el hilo. El teclado ejecuta matchbox-keyboard como se planteo anteriormente, mientras que los hijos de manejo_control se ejecutan posterior al teclado virtual. El menu no se planteo en un hilo independiente porque el mainLoop() de tkinter retendría de cualquier modo la ejecución del mismo.

## 11. Cronograma: Diagrama de Gantt

Para el desarrollo de este proyecto se planteo un cronograma inicial de tiempos que nos permitió gestionar el desarrollo de modo que se cumplieran las fechas acordadas de entrega. Para el planteamiento del cronograma se centró en el desarrollo de los módulos de software descritos con anterioridad. Este desarrollo se planteo originalmente como el desarrollo de módulos independientes cuyas pruebas se realizaron por separado. Con esta idea, se siguió el siguiente diagrama:

| Actividad | Inicio | Fin | 15-nov-23 | 20-nov-23 | 21-oct-23 | 22-oct-23 | 23-nov-23 | 24-nov-23 | 28-nov-23 | 29-nov-23 | 02-dic-23 | 03-dic-23 | 04-dic-23 | 05-dic-23 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Investigaciones Previas** (sistema operativo, material de circuito, bibliotecas relacionadas,etc.) | 15-nov-23 | 20-nov-23 | | | | | | | | | | | | |
| **Planteamiento inicial**: definición de módulos, idea general de proyecto, cronograma | 21-nov-23 | 22-nov-23 | | | | | | | | | | | | |
| **Reproducción de animación:** generación de animación y carga en programa | 23-nov-23 | 23-nov-23 | | | | | | | | | | | | |
| **Set de rom precargadas**: carga de roms guardadas, creación de lista de juegos, menú de selección | 24-nov-23 | 28-nov-23 | | | | | | | | | | | | |
| **Detección de USB y carga de nuevas ROM:** detección de conexiones, enlistado de juegos en memoria, comparación con juegos almacenados en sistema, copia de juegos nuevos | 29-nov-23 | 02-dic-23 | | | | | | | | | | | | |
| **Control completo de consola usando gamepad:** creación de circuito, mapeado de botones a pines, mapeado de botones (software), mapeado de botones en mednafen | 02-dic-23 | 04-dic-23 | | | | | | | | | | | | |
| **Arranque directo a consola:** archivos de arranque | 04-dic-23 | 04-dic-23 | | | | | | | | | | | | |
| **Documentación:** Reporte escrito, repositorio github y configuración automatizada | 04-dic-23 | 03-dic-23 | | | | | | | | | | | | |
| **Presentación final:** demostración física del proyecto | 05-dic-23 | 05-dic-23 | | | | | | | | | | | | |

Figura 3: *Cronograma general de desarrollo*

Cada punto aquí listado se planteó teniendo en cuenta la idea general de proyecto planteada por los requisitos de proyecto y el diseño ideado a comienzos del mismo. Así mismo, podemos detallar un poco más en el cronograma de los módulos con mayor tiempo requerido [Figura 4].

## 12. Conclusión

El desarrollo de sistemas embebidos es un proceso que requiere de pleno análisis y dominio del tema. Como puede entenderse, el desarrollo de este proyecto requirió plantear problemáticas distintas y abarcar más posibilidad de escenarios que los que se esperaban. Por ejemplo, el detector de USB se planteo para detectar cualquier USB, ya fuera que se detectara como disk o partition. De cualquier modo, existieron problemáticas que pudieron atenderse a partir del precepto de que un sistema embebido cumple con un propósito específico. De aquí que pudiéramos emplear rutas absolutas pues sin importar que un usuario u otro ejecute el proyecto, este está planeado para que opere homogéneamente, por lo que las rutas son homogéneas. Esta poca variación entre sistemas nos permitió generalizar distintos escenarios para evitar problemáticas de las que se tenían conflictos.

Así mismo, se reconoce la necesidad de comprender más a fondo el sistema con el que se opera, pues existen

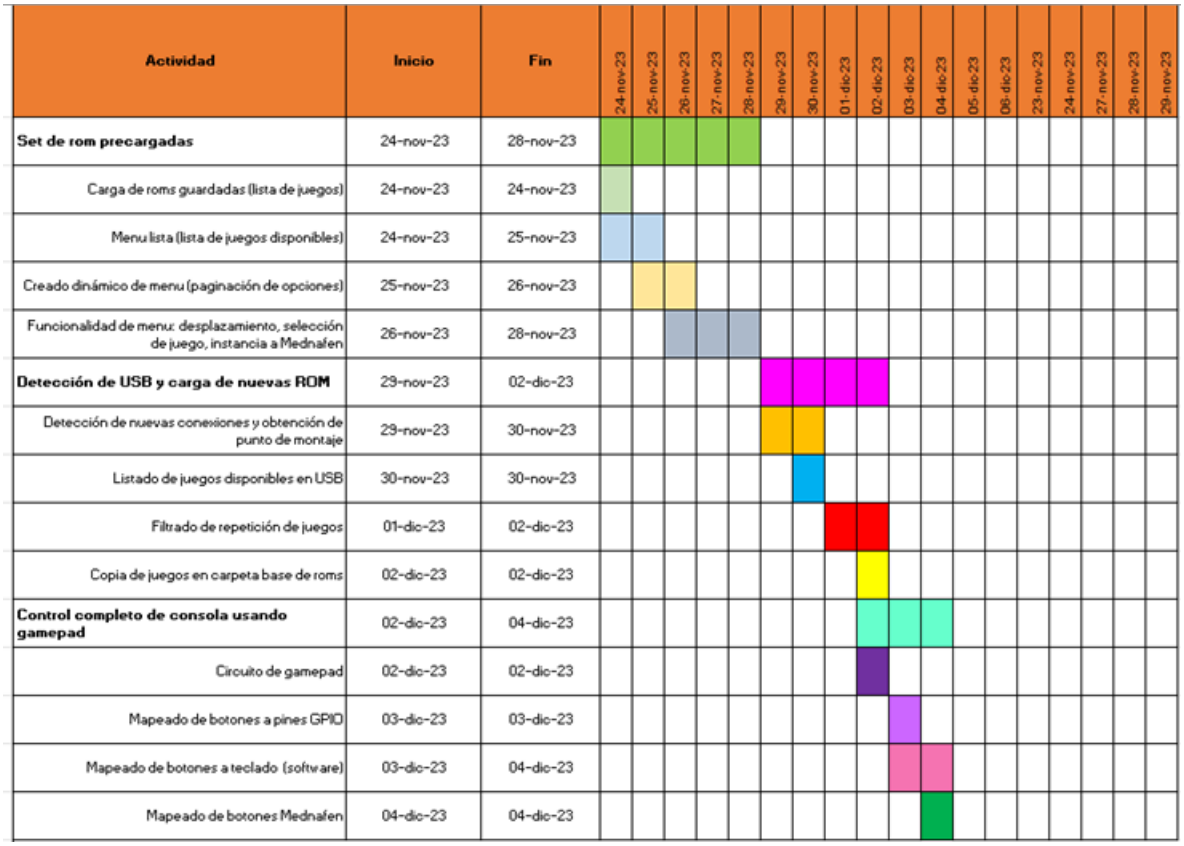


Figura 4: *Cronograma específico de módulos*

configuraciones necesarias para los proyectos que no pueden ser realizados sin acceder a la documentación de los mismos. Lo mismo sucede con los módulos empleados por el sistema. Por ejemplo, Mednafen requiere de la configuración de algunas variables como video.fs 1 para establecer en pantalla completa, o sound.device hw=2. Este último viene indicado en la propia documentación de Mednafen dado a que existe el error recurrente donde el programa no opera correctamente con los dispositivos de audio en Linux. Estos cambios no se sabe que son necesarios hasta que se investiga más a fondo la documentación.

## Referencias

[1] Mednfen Bettle (2022a). Introduction to Mednafen. Recuperado de: `https://mednafen.github.io`. Consultado el 04 de Diciembre de 2023.

[2] Mednfen Bettle (2022b). Troubleshooting and Common Solutions. Recuperado de: `https://mednafen.github.io/documentation/`. Consultado el 04 de Diciembre de 2023.

[3] PyPI (06 de Febrero del 2023). RPi.GPIO 0.7.1. Recuperado de: https://pypi.org/project/RPi.GPIO/. Consultado el 16 de septiembre de 2022.

[4] Python Software Foundation (2023). tkinter. Recuperado de: https://docs.python.org/es/3/library/tkinter.htm Consultado el 04 de Diciembre de 2023.

[5] Raspberry Pi Foundation (23 de Agosto del 2023). GPIO and the 40-pin header. Recuperado de: https://www.raspberrypi.com/documentation/computers/raspberry-pi.html#gpio-and-the-40-pin-header. Consultado el 16 de septiembre de 2023.

[6] RaspberryPi, F. (2019). Raspberry pi 4. Recuperado de: www.raspberrypi.com/products/raspberry-pi-4-model-b/ Consultado el 04 de Diciembre de 2023.