# Creación de una consola retro con la Raspberry Pi 4

María Emilia Ramírez Gómez

## 1 Objetivo

Configurar una Raspberry Pi 4 como una consola de videojuegos capaz de emular las plataformas NES, SNES y Game Boy Advance. El sistema incluirá una interfaz gráfica desarrollada en Python, diseñada para ser controlada mediante los Joy-Con de Nintendo.

## 2 Introducción

En esta práctica, convertiremos una Raspberry Pi 4 en una consola de videojuegos capaz de emular sistemas como Nintendo Entertainment System, Super Nintendo y Game Boy Advance. Para ello, desarrollaremos una aplicación que integrará el emulador Mednafen con una interfaz gráfica, permitiendo seleccionar y ejecutar videojuegos de manera intuitiva.

Además, implementaremos funcionalidades para conectar un par de controles Joy-Con, eliminando la necesidad de usar teclado o mouse para interactuar con el sistema. También gestionaremos la incorporación de nuevos juegos al catálogo de la consola mediante la detección automática de dispositivos USB conectados.

Asimismo, se explicarán las configuraciones necesarias para personalizar el proceso de arranque del sistema operativo, de modo que se muestre una imagen estática de forma silenciosa (sin mensajes en pantalla). Y una vez que todos los servicios estén cargados, nuestro programa se ejecute automáticamente.

## 3 Antecedentes

**Plymouth**, un proyecto desarrollado por fedora para personalizar una animación durante el proceso de arranque del sistema operativo. Este paquete proporciona temas predefinidos y nos permite desarrollar los propios. [4, 9]

**Getty** es un programa que gestiona la consola virtual (tty) que se encarga de mostrar el mensaje de inicio de sesión, el proceso de autenticación e inicio de sesión. [1]

Un **Emulador** es una aplicación de software que permite que un sistema funcione como si fuera otro, facilitando la compatibilidad con hardware o software diferente. [6] En el caso de videojuegos, los emuladores permiten ejecutar juegos diseñados para consolas específicas en otros dispositivos, como computadoras, imitando el comportamiento del hardware original.

**Mednafen** es un emulador multisistema portátil que utiliza OpenGL y SDL. Este programa se maneja a través de comandos en la terminal y es compatible con varias consolas de Nintendo, Sega, Sony, Atari, etc. [7]

La **Raspberry Pi 4** es una mini computadora que trabaja con un procesador ARM de varios núcleos desarrollada por la Raspberry Pi Foundation. Esta cuenta con Bluetooth 5.0, lo que nos permite conectar de manera inalámbrica a dispositivos periféricos. Por otro lado, la Raspberry Pi es compatible con OpenGL ES 3.1, una API gratuita y multiplataforma para renderizar gráficos avanzados en 2D y 3D en sistemas embebidos y móviles [8, 10]

## 4 Materiales

- Una Raspberry Pi 4 con 4 GB de RAM
- Una USB
- Un cable micro HDMI a HDMI
- Una fuente de alimentación regulada a 5V y al menos 2 amperios de salida
- Un par de controles Joy-Con de Nintendo

Importante

Los controles Joy-Con no son estrictamente necesarios, en caso de utilizar otro gamepad se tendrán que realizar ajustes en el código.

# 5 Configuración para convertir la Raspberry Pi 4 en una consola retro

Antes de comenzar, es importante asegurarse de que la Raspberry Pi 4 cuente con la versión Lite del sistema operativo Raspberry Pi OS. Para este proyecto, se utilizó la versión publicada el 4 de julio de 2024 (`2024-07-04`), disponible en el siguiente enlace: https://downloads.raspberrypi.com/raspios_lite_armhf/images/raspios_lite_armhf-2024-07-04/.

Los archivos necesarios para configurar la Raspberry Pi como una consola retro están alojados en el repositorio de GitHub https://github.com/MariaEmiliaRG/arcadiax. Dicho repositorio debe ser clonado en el directorio `home`. Este repositorio debe ser clonado en el directorio texttt/home. Para hacerlo, primero es necesario instalar Git con el siguiente comando: `sudo apt install git`. Luego, el repositorio se puede clonar ejecutando el siguiente comando: `git clone https://github.com/MariaEmiliaRG/arcadiax.git` Es importante señalar que todo el desarrollo de este proyecto se realizó bajo la licencia MIT.

En caso de no querer clonar el repositorio, descarga los archivos de la carpeta imgs donde se encuentra el contenido multimedia para este proyecto y asegúrate de modificar las referencias a estos archivos más adelante.

## 5.1 Configuración de la imagen y sonido de arranque

Para configurar una imagen de arranque personalizada utilizaremos el paquete `plymouth`. Este proceso mostrará una imagen estática durante el arranque del sistema. Primero, es necesario instalar el paquete `plymouth` y sus temas con el siguiente comando:

```
sudo apt-get install plymouth plymouth-themes
```

Enseguida, acceder al directorio `/usr/share/plymouth/themes/` y crear una carpeta. En este caso se llamará `arcadiax`.

```
sudo mkdir /usr/share/plymouth/themes/arcadiax
```

Dentro de la carpeta arcadiax crea el archivo `arcadiax.plymouth` donde colocaremos las siguientes líneas:

```
[Plymouth Theme]
Name=arcadiax
Description=Un tema personalizado para Plymouth
ModuleName=script

[script]
ImageDir=/usr/share/plymouth/themes/arcadiax
ScriptFile=/usr/share/plymouth/themes/arcadiax/arcadiax.script
```

Donde en la primera sección `Plymouth Theme` describimos el nuevo tema, y en la segunda sección, `script`, se define la ubicación de la imagen que se mostrará durante el proceso de arranque y el script que contiene las instrucciones para mostrar dicha imagen. El script es el siguiente:

```
image = Image("arcadiax-boot.png");

# Obtener las dimensiones de la pantalla y la imagen
screen_width = Window.GetWidth();
screen_height = Window.GetHeight();
image_width = image.GetWidth();
image_height = image.GetHeight();

# Escalar la imagen para que se ajuste a la pantalla manteniendo la proporcin
scale_factor = min(screen_width / image_width, screen_height / image_height);

# Redimensionar la imagen manteniendo la proporcin
image.SetWidth(image_width * scale_factor);
image.SetHeight(image_height * scale_factor);

# Calcular las posiciones para centrar la imagen redimensionada
pos_x = Window.GetWidth()/2 - image.GetWidth()/2;
pos_y = Window.GetHeight()/2 - image.GetHeight()/2;
```

```
# Crear el sprite con la imagen escalada y ajustada
sprite = Sprite(image);
sprite.SetX(pos_x);
sprite.SetY(pos_y);

# Establecer la opacidad del sprite
sprite.SetOpacity(1);
```

Ambos archivos, así como la imagen del proceso de arranque se encuentran en la carpeta `Playmouth` del repositorio. Por último, para establecer el tema como predeterminado se debe colocar la siguiente instrucción en línea de comandos: `sudo plymouth-set-default-theme -R arcadiax`.

Posteriormente, para ocultar los mensajes de arranque, es necesario editar el archivo `/boot/firmware/cmdline.txt` agregando las siguientes opciones al final de la línea existente (sin crear líneas nuevas): `quiet splash`. La opción `quiet` se encarga de ocultar los mensajes que suelen aparecer durante el proceso de arranque, mientras que `splash` activa la pantalla de arranque visual para que se pueda visualizar el tema de Plymouth [2, 11]. El contenido del archivo debe verse de la siguiente manera:

```
console=serial0,115200 console=tty1 root=PARTUUID=0b0b669c-02 rootfstype=ext4 fsck.repair=yes rootwait quiet
    splash
```

Por otro lado, para deshabilitar la pantalla arcoíris que aparece al inicio del sistema se debe editar el archivo `/boot/firmware/config.txt` agregando las siguientes líneas al final del archivo:

```
disable_splash=1
framebuffer_width=1920
framebuffer_height=1080
```

Una vez realizadas todas las modificaciones, guarda los cambios en cada archivo y reinicia el sistema para aplicar los cambios. Ambos archivos de configuración se encuentran en la carpeta conf del repositorio.

El siguiente paso es configurar el sonido de arranque, y esto se logra mediante la creación de un servicio que se ejecutará al iniciar el sistema. Primero instalaremos Pulse Audio para gestionar la entrada y salida de audio en el sistema.

```
sudo apt install pulseaudio
systemctl --user enable pulseaudio
systemctl --user start pulseaudio
```

De igual manera, nos aseguraremos de que la salida de audio sea la correcta ejecutando `sudo raspi-config` en la terminal. Seleccionaremos la opción `System Options`, luego `Audio` y por último `HDMI`. Enseguida instalaremos el paquete FFmpeg con la instrucción `sudo apt install ffmpeg`.

En seguida, crearemos un script en el directorio `/home/arcadiax/scripts` que contendrá la instrucción para la reproducción del audio. El contenido de este será:

```
#!/bin/bash
# Reproduce el archivo de audio en el arranque
ffmpeg -i /home/emilia/arcadiax/imgs/twinkle.mp3 -f pulse "default"
```

En caso de contar con otro archivo de audio, modificar la ruta de este. Luego, crearemos un nuevo servicio. Para ello debemos crear un archivo en `/etc/systemd/system/` con el nombre de `play-boot-audio.service` donde su contenido será el siguiente:

```
[Unit]
Description=Reproducir audio al iniciar el sistema
After=default.target

[Service]
Type=oneshot
ExecStart=/bin/bash -c '/home/emilia/arcadiax/scripts/play-boot-audio.sh'
User=emilia
Group=emilia
Environment=PULSE_RUNTIME_PATH=/run/user/1000/pulse/

[Install]
WantedBy=default.target
```

En la sección `[Unit]`, se incluye una breve descripción del servicio. Además, con `After=default.target` especifica que el servicio debe ejecutarse solo después de que `default.target` se encuentre activa[3]

La sección `[Service]` contiene la configuración de cómo se ejecutará el servicio. La opción `Type=oneshot` indicamos que el servicio se ejecuta una vez y finaliza sin quedarse en ejecución. Por otro lado, también indicamos el comando que se ejecutará con `ExecStart=/bin/bash -c '/home/emilia/arcadiax/scripts/play-boot-audio.sh'`,

en caso de haber guardado el script en otro lugar, es importante cambiar la ruta. Las opciones `User=emilia` y `Group=emilia` definen que el servicio se ejecutará con los privilegios del usuario y grupo emilia. No olvides colocar el nombre correspondiente al usuario que se haya especificado durante la instalación del sistema operativo. Por último, se configura una variable de entorno para garantizar que el servicio pueda acceder correctamente al servidor de sonido PulseAudio, `Environment=PULSE_RUNTIME_PATH=/run/user/1000/pulse/`. [5, 3]

Finalmente, la sección `[Install]` con WantedBy=default.target, se creará una dependencia débil con `WantedBy=default.ta` lo que significa que no es obligatorio que funcione. [3] El último paso es ejecutar las siguientes instrucciones en la terminal:

```
sudo systemctl enable play-boot-audio.service
sudo systemctl enable play-boot-audio.service
```

Enseguida es necesario configurar la opción de `autologin`. Esto nos permitirá ingresar al sistema sin necesidad de colocar la contraseña del usuario. Esto se logra a través de modificar el archivo de configuración de `getty /etc/systemd/system/getty@tty1.service.d/autologin.conf`. El contenido de dicho archivo deberá ser el siguiente:

```
/etc/systemd/system/getty@tty1.service.d/autologin.conf
[Service]
ExecStart=
ExecStart=-/sbin/agetty -o '-p␣-f␣--␣\\u' --noclear --autologin emilia %I $TERM
```

Enseguida reiniciamos el servicio con

```
sudo systemctl daemon-reload
sudo systemctl restart getty@tty1
```

Con todas estas configuraciones, al reiniciar deberíamos observar los cambios.

## 5.2 Instalación de los paquetes para controlar los Joy-Con

Para poder utilizar el par de Joy-Con como un solo dispositivos nos apoyaremos de dos proyectos: dkms-hid-nintendo y joycond; El primero es un módulo de kernel que mejora la compatibilidad de Linux con los dispositivos de Nintendo, por otro lado, `joycond` es un demonio que facilita el uso de ambos controles como un único dispositivo al presionar al mismo tiempo los gatillos (ZR y ZL). Para la instalación de ambos se deben seguir los siguientes pasos:

```
git clone https://github.com/nicman23/dkms-hid-nintendo
cd dkms-hid-nintendo

sudo dkms add .
sudo dkms build nintendo -v 3.2
sudo dkms install nintendo -v 3.2

git clone https://github.com/DanielOgorchock/joycond.git
cd joycond
sudo apt install libevdev-dev libudev-dev cmake
cmake .
sudo make install
sudo systemctl enable --now joycond
```

## 5.3 Instalación del emulador de las consolas SNES, NES y Gameboy Advance

Mednafen es un emulador multi-sistema que permite jugar a juegos de consolas clásicas en diversas plataformas, y para este proyecto es conveniente porque no cuenta con interfaz gráfica por defecto, ya que está diseñado para ejecutarse desde la línea de comandos. Su instalación es bastante sencilla, en la terminal es necesario colocar `sudo apt install mednafen`. Para corroborar que el proceso ha sido exitoso, podemos probar utilizando un juego que se encuentra en la carpeta roms del repositorio de GitHub. Para ello, en la terminal utilizamos el comando `mednafen` seguido de la ruta a la ubicación del juego: `mednafen LEYEND-OF-ZELDA.nes`.

Es probable que al iniciar el juego este no produzca ningún sonido y que no se muestre en modo pantalla completa. Para corregirlo se deben modificar las siguientes variables del archivo de configuración `~/.mednafen/mednafen.cfg`:

```
sound.driver sdl
video.fs 1
```

Así mismo, al presionar la tecla `F1`, se abrirá una pantalla con una lista de todas las funciones disponibles así como su combinación de teclas para hacer uso de ellas. Entre estas se encuentra el mapeo de los controles con `alt + shift + 1`.

## 5.4 Desarrollo de una interfaz para la integración del emulador, los controles y videojuegos

El desarrollo de la consola se divide en módulos con la intención de mejorar la organización, la comprensión y el mantenimiento del código. Todos los archivos se encuentran en la carpeta modules del repositorio.

- **Bluetooth**: Conjunto de funcionalidades que facilitan la conexión de dispositivos a través de Bluetooth, interactuando con la terminal de comandos `bluetoothctl` para gestionar dispositivos emparejados y configurar conexiones de manera eficiente.
- **JoyCons**: Módulo responsable de gestionar la conexión y desconexión de los controles para que estos sean detectados y puedan interactuar adecuadamente con el sistema.
- **USBManager**: Métodos diseñados para detectar la inserción de dispositivos USB, así como para gestionar los procesos de montaje y desmontaje de estos dispositivos en el sistema, asegurando una correcta integración.
- **ROMSManager**: Funciones encargadas de la gestión de juegos en la consola, permitiendo agregar nuevos juegos a la biblioteca de la consola, y asegurando que no se repitan títulos previamente añadidos.
- **Interface**: Métodos destinados a la creación y gestión de la interfaz gráfica del proyecto.
- **Arcadiax**: Clase principal que integra y coordina los módulos anteriores, proporcionando la funcionalidad completa del sistema para el correcto funcionamiento de la consola de videojuegos.

Para ello es necesario descargar los siguientes módulos de python: sudo apt install python3-pygame python3-pexpect python3-uinput python3-pyudev

### 5.4.1 Bluetooth

Esta clase permite interactuar con dispositivos Bluetooth usando la herramienta bluetoothctl a través de métodos para emparejar, conectar, desconectar y eliminar dispositivos Bluetooth utilizando su dirección MAC. Para lograr esto, se emplea el módulo `pexpect`.

`pexpect` se utiliza para manejar sesiones interactivas con programas de terminal, en este caso, bluetoothctl, simulando la entrada y salida de comandos para garantizar que cada paso se complete correctamente antes de proceder al siguiente (fase emparejamiento y conexión).

```
import pexpect
import time

class Bluetooth:

    def __init__(self):
        pass

    def initSession(self):

        self.session = pexpect.spawn("bluetoothctl", encoding="utf-8")
        self.session.expect("#")

        return

    def connectDevice(self, macAddress):
        self.initSession()

        self.session.sendline(f"pair␣{macAddress}")
        try:
            self.session.expect("Pairing␣successful", timeout=20)
        except pexpect.exceptions.TIMEOUT:
            self.session.sendline("exit")
            self.session.close()
            return 0

        time.sleep(2)
        self.session.sendline(f"connect␣{macAddress}")
        try:
            self.session.expect("Connection␣successful", timeout=20)
        except pexpect.exceptions.TIMEOUT:
            return 0
```

```
        self.close()

        return 1

    def disconnectDevice(self, macAddress):
        self.initSession()
        self.session.sendline(f"disconnect␣{macAddress}")
        index = self.session.expect(["Successful␣disconnected", "Failed␣to␣disconnect", pexpect.EOF, pexpect.
            TIMEOUT])
        self.close()
        if index == 0:
            return 1
        else:
            return 0

    def removeDevice(self, macAddress):
        self.initSession()
        self.session.sendline(f"remove␣{macAddress}")
        index = self.session.expect(["Device␣has␣been␣removed", "Failed␣to␣remove", pexpect.EOF, pexpect.TIMEOUT
            ])
        self.close()
        if index == 0:
            return 1
        else:
            return 0

    def close(self):
        self.session.sendline("exit")
        self.session.close()

        return
```

### 5.4.2 JoyCons

Enseguida, tenemos la clase JoyCons que maneja la conexión y desconexión de los mandos Joy-Con utilizando el módulo anterior `Bluetooth`, `pygame` y `subprocess`. Cada control cuenta con una dirección MAC, la cual hay que conocer con anticipación para poder realizar la vinculación de estos dispositivos.

`bluetoothctl` cuenta con la funcionalidad de escanear dispositivos Bluetooth cercanos y de este modo identificar los dispositivos disponibles para conectarse. Así mismo, se pueden listar los dispositivos que se han encontrado. Para identificar las direcciones MAC de ambos controles se utilizará un script que hace uso de ambas funciones, obteniendo así las direcciones de los dispositivos que por nombre hagan referencia a `Joy-Con`. Este script es llamado desde el método `getMacJoyCons`.

```
#!/bin/bash

bluetoothctl power on > /dev/null &
bluetoothctl scan on > /dev/null &
SCAN_PID=$!

macJoyConL=""
macJoyConR=""

while [ -z "$macJoyConR" ] || [ -z "$macJoyConL" ]; do
    macJoyConR=$(bluetoothctl devices | grep "Joy-Con␣(R)" | awk '{print␣$2}')
    macJoyConL=$(bluetoothctl devices | grep "Joy-Con␣(L)" | awk '{print␣$2}')
    sleep 1
done

kill $SCAN_PID

echo "{\"macJoyConR\":␣\"$macJoyConR\",␣\"macJoyConL\":␣\"$macJoyConL\"}"
```

La clase incluye métodos para conectar, desconectar y eliminar las direcciones MAC de los controles, con el objetivo de facilitar que la consola de videojuegos pueda interactuar con otros dispositivos similares. Esto porque los controles Joy-Con comparten el mismo nombre, lo que dificulta diferenciarlos. Además, aunque los dispositivos no estén físicamente cerca, el sistema operativo almacena las direcciones MAC de todos los dispositivos previamente vinculados. Esta acumulación puede complicar la identificación de dispositivos nuevos al intentar emparejarlos y conectarlos. Por esta razón, al desconectar un control, es más práctico eliminar su registro, facilitando la detección y conexión de dispositivos en el futuro.

```
import subprocess
import json
import Bluetooth
import pygame
import time

class JoyCons:

    def __init__(self):
        self.pathScriptGetMac = "../scripts/bluetooth-get-mac-joycons.sh"
        self.joyconsMac = {}
        self.joystick = None
        self.joyconsNames = ["Nintendo Switch Combined Joy-Cons","Nintendo Switch Right Joy-Con","Nintendo
            Switch Left Joy-Con"]
        self.joycon1 = None
        pygame.init()

    def getMacJoyCons(self):
        result = subprocess.run([self.pathScriptGetMac], capture_output=True, text=True, check=True)
        self.joyconsMac = json.loads(result.stdout)

        print(self.joyconsMac)

        return

    def connectJoyCons(self):
        self.getMacJoyCons()
        bt = Bluetooth.Bluetooth()

        pygame.mixer.init()
        pygame.mixer.music.load('../imgs/twinkle.mp3')

        for _, mac in self.joyconsMac.items():
            bt.connectDevice(mac)

        pygame.mixer.music.play()
        return

    def disconnectJoyCons(self):
        # we remove the macs since every right and left joycon have the same name
        # it's eassier identifying them by the first time instead of trying connection with every single mac
            address
        self.getMacJoyCons()


        bt = Bluetooth.Bluetooth()

        for _ , mac in self.joyconsMac.items():
            bt.disconnectDevice(mac)
            bt.removeDevice(mac)

        return

    def changeMap(self, numJoyCons):
        #TODO modificar archivos de configuracin con el mapeo de los controles
        pass

    def countJoyCons(self):
        total = self.joystick.get_count()
        self.changeMap(total)

        return total

    def initJoyCon1(self):
        while pygame.joystick.get_count() <= 0:
            self.joystick = pygame.joystick.init()
            print(pygame.joystick.get_count())
            time.sleep(5)

        self.joycon1 = pygame.joystick.Joystick(0)
        self.joycon1.init()

        return
```

Por último, el método `initJoyCon1` tiene como propósito inicializar el primer Joy-Con que se detecte conectado al sistema utilizando la biblioteca pygame. Esto nos permitirá que podamos interactuar con el sistema en cuanto se detecte un dispositivo.

### 5.4.3 USBManager

Un componente clave de la consola es poder integrar nuevos juegos de manera automática. Esto lo vamos a lograr a través de la detección de USB que contengan los juegos compatibles con el emulador. `USBManager` está diseñada para gestionar automáticamente dispositivos USB en un sistema Linux, utilizando las herramientas del sistema y la biblioteca pyudev. Su objetivo principal es detectar, montar y gestionar particiones USB.

El método principal es `usbDetection` el cual utiliza un bucle para escuchar continuamente los eventos del sistema relacionados con dispositivos de almacenamiento. Cuando detecta un evento de tipo "add", que indica la conexión de un nuevo dispositivo USB, monta automáticamente la partición correspondiente utilizando el método `autoMount`. Después, utiliza `getMountPoint` para recuperar el punto de montaje del dispositivo recién conectado, asegurándose de que el sistema puede acceder a los datos del USB.

```
import os
import pyudev
import subprocess as sp

class USBManager:
    def __init__(self):
        self.context = pyudev.Context()
        self.monitor = pyudev.Monitor.from_netlink(self.context)
        self.monitor.filter_by(subsystem="block", device_type="partition")
        self.flag = True

        return

    def autoMount(self, path):
        args = ["udisksctl", "mount", "-b", path]
        sp.run(args)

        return

    def autoUnmount(self, path):
        args = ["udisksctl", "unmount", "-b", path]
        sp.run(args)

        return

    def getMountPoint(self, path):
        args = ["findmnt", "-unl", "-S", path]
        cp = sp.run(args, capture_output=True, text=True)
        out = cp.stdout.split("␣")[0]

        return out

    def usbDetection(self):
        mountPoint = None

        while self.flag:

            device = self.monitor.poll(timeout=1)
            if device is None:
                continue
            else:
                action = device.action

                if action != "add":
                    print(self.flag)
                    continue

                devicePath = "/dev/" + device.sys_name
                self.autoMount(devicePath)
                mountPoint = self.getMountPoint(devicePath)
                self.flag = False

        return mountPoint
```

### 5.4.4 ROMSManager

Con la clase anterior, podemos identificar la ruta al directorio que podría tener nuevos juegos. Sin embargo, es necesario filtrar los archivos que se añadirán, ya que el emulador no es compatible con todas las consolas. Además, tenemos que evitar incorporar juegos duplicados para mantener el sistema organizado. En la carpeta roms, se encuentra un archivo JSON que especifica las consolas compatibles junto con las extensiones correspondientes a sus juegos. Si deseamos ampliar el catálogo de consolas en el sistema, basta con agregar un nuevo elemento al archivo, siguiendo el formato establecido (siempre y cuando la consola sea compatible con el emulador).

```
{
    "Game␣Boy␣Advance": [".gba"],
    "Super␣Nintendo␣Entertainment␣System": [".smc", ".sfc"],
    "Nintendo␣Entertainment␣System": [".nes"]
}
```

ROMSManager se encarga de gestionar las ROMs de videojuegos, organizándolas, filtrándolas. Una de sus funciones está en estandarizar el nombre de los videojuegos que ingresan. changeROMName elimina caracteres no deseados, como paréntesis o corchetes, y transforma el nombre sustituyendo espacios y caracteres especiales por guiones y convirtiéndolo a mayúsculas. Esto asegura que los nombres de los archivos sean fáciles de manejar, ya que las páginas donde podemos obtener estos juegos a veces suelen colocar entre corchetes o paréntesis información adicional (versión, lenguaje, etc).

Para gestionar estos elementos se deben crear funciones que permitan verificar si el archivo ya existe, así como la copia de estos a la carpeta correspondiente. Por otro lado, se necesita un método que examine un directorio y devuelve una lista de archivos que coincidan con las extensiones válidas para las consolas soportadas.

Finalmente, el método getROMSGroupByConsole organiza las ROMs almacenadas en el directorio principal agrupándolas según su consola. Utiliza el archivo JSON de extensiones para determinar a qué consola pertenece cada archivo, devolviendo un diccionario donde las claves son los nombres de las consolas y los valores son listas de las ROMs asociadas.

```
import os
import re
import shutil
import json

class ROMAlreadyExistsException(Exception):
    """Excepcin lanzada cuando la ROM ya existe en el directorio."""
    pass

class ROMSManager:
    def __init__(self):
        self.ROMSDir = "../roms"
        if not os.path.exists(self.ROMSDir):
            os.makedirs(self.ROMSDir)

        with open(os.path.join(self.ROMSDir,"ROMSExt.json"), "r") as file:
            self.ROMSExtensions = json.load(file)

        return

    def changeROMName(self, fileName):
        name, extension = os.path.splitext(fileName)
        name = re.sub(r'[\(\[].*?[\)\]]', '', name)
        name = re.sub(r'[^A-Za-z0-9␣]', '␣', name)
        name = name.split()
        name = "-".join(name)
        name = name.upper()

        return name + extension

    def ROMExist(self, fileName):
        return os.path.exists(os.path.join(self.ROMSDir, fileName))

    def copyROM(self, originPath, fileName):
        newFileName = self.changeROMName(fileName)
        if self.ROMExist(newFileName):
            raise ROMAlreadyExistsException(f"El␣videojuego␣{fileName}␣ya␣existe.")
        try:
            shutil.copy(os.path.join(originPath, fileName), os.path.join(self.ROMSDir, newFileName))
        except Exception as e:
            raise
```

```
        return

    def getROMSFromADir(self, dirPath):

        ROMSFiles = []

        for fileName in os.listdir(dirPath):
            filePath = os.path.join(dirPath, fileName)
            if os.path.isfile(filePath):
                for extensions in self.ROMSExtensions.values():
                    if fileName.lower().endswith(tuple(extensions)):
                        ROMSFiles.append(fileName)
                        break

        return ROMSFiles

    def getROMSGroupByConsole(self):
        ROMSFiles = {}

        for fileName in os.listdir(self.ROMSDir):
            filePath = os.path.join(self.ROMSDir, fileName)

            if os.path.isfile(filePath):
                for console in self.ROMSExtensions:
                    if fileName.lower().endswith(tuple(self.ROMSExtensions[console])):
                        if console not in ROMSFiles:
                            ROMSFiles[console] = []
                        ROMSFiles[console].append(fileName)
                        break

        return ROMSFiles
```

### 5.4.5 Interface

Este módulo está diseñado para dibujar la interfaz gráfica utilizando la biblioteca pygame. Al instanciarla, esta establece la configuración básica de la pantalla, adaptándola a la resolución del dispositivo. Las funcionalidades principales es la creación de rectángulos y texto, que se utilizan para representar los botones con los que se podrá interactuar con el menú.

Las características de los botones serán definidas en un archivo JSON donde se incluirá su posición en el eje y (ya que se buscará que se encuentre centrado). El color de este, el texto que tendrá dentro el rectángulo y si ha sido seleccionado por el usuario. El archivo se ve de la siguiente manera:

```
{
    "console": {
        "y": 70,
        "backgroundColor": [216, 187, 255],
        "borderColor": [216, 187, 255],
        "borderWidth": 10,
        "text": "CONSOLA",
        "fontSize": "big",
        "textColor": [53, 40, 79],
        "selected": 1
    },
    "game": {
        "y": 290,
        "backgroundColor": [216, 187, 255],
        "borderColor": [216, 187, 255],
        "borderWidth": 10,
        "text": "VIDEOJUEGO",
        "fontSize": "big",
        "textColor": [53, 40, 79],
        "selected": 0
    },
    "play": {
        "y": 510,
        "backgroundColor": [216, 187, 255],
        "borderColor": [216, 187, 255],
        "borderWidth": 10,
        "text": "JUGAR",
```

```
        "fontSize": "big",
        "textColor": [53, 40, 79],
        "selected": 0
    },
    "controls": {
        "y": 655,
        "backgroundColor": [53, 40, 79],
        "borderColor": [216, 187, 255],
        "borderWidth": 10,
        "text": "APAGAR",
        "fontSize": "big",
        "textColor": [216, 187, 255],
        "selected": 0
    },
    "consoleOptions": {
        "y": 145,
        "backgroundColor": [53, 40, 79],
        "borderColor": [216, 187, 255],
        "borderWidth": 10,
        "text": "consola",
        "fontSize": "small",
        "textColor": [216, 187, 255],
        "selected": 0
    },
    "gameOptions": {
        "y": 365,
        "backgroundColor": [53, 40, 79],
        "borderColor": [216, 187, 255],
        "borderWidth": 10,
        "text": "videojuego",
        "fontSize": "small",
        "textColor": [216, 187, 255],
        "selected": 0
    }
}
```

Otro punto importante del módulo es su capacidad para gestionar la visibilidad de la interfaz gráfica. Esto porque el emulador utiliza el controlador de video durante su funcionamiento, y no es posible que ambos procesos lo estén utilizando al mismo tiempo. Para esto, se emplean las funciones `hideDisplay` y `showDisplay`, que manipulan la variable de entorno `SDL_VIDEODRIVER` para cambiar entre controladores.

`hideDisplay` cambia la variable de entorno `SDL_VIDEODRIVER` a `dummy`, lo que indica a `pygame` que no utilice un controlador de video permitiendo la liberación de recursos gráficos para que mednafen pueda hacer uso de ellos. Por otro lado `showDisplay` hace lo contrario, restablece el controlador a `kmsdrm`, volviendo a habilitar la visualización de la interfaz gráfica en la pantalla del dispositivo. En este caso, es necesario llamar a `pygame.display.init` para restablecer la configuración.

```
import pygame
import json
import time
import os

class Interface:
    def __init__(self):
        os.environ["SDL_VIDEODRIVER"] = "kmsdrm"
        pygame.init()
        pygame.display.init()
        info = pygame.display.Info()
        self.WIDTH, self.HEIGHT = info.current_w, info.current_h

        self.screen = pygame.display.set_mode((self.WIDTH, self.HEIGHT))

        self.DARK_PURPLE = (53, 40, 79)
        self.LIGHT_PURPLE = (216, 187, 255)
        self.YELLOW = (255, 222, 89)

        self.BUTTON_WIDTH = int(self.WIDTH * 0.75)
        self.BUTTON_HEIGHT = int(self.HEIGHT * 0.09375)

        self.FONT = pygame.font.Font(None, 50)
        self.SMALL_FONT = pygame.font.Font(None, 30)

        with open("mainMenuButtons.json", "r", encoding='utf-8') as jsonFile:
```

```
        self.mainMenuButtons = json.load(jsonFile)

    with open("controlsMenuButtons.json", "r", encoding='utf-8') as jsonFile:
        self.controlsMenuButtons = json.load(jsonFile)

    self.imgBackground = pygame.image.load("../imgs/arcadiax-background.png")

    self.imgBackground = pygame.transform.scale(self.imgBackground, (self.WIDTH, self.HEIGHT))

    return

def getY(self, y):
    return int(y * self.HEIGHT / 800)

def drawRect(self, y, width, height, buttonColor, borderColor=None, borderWidth=0):
    x = (self.WIDTH - width) // 2

    rect = pygame.Rect(x, y, width, height)
    if borderColor and borderWidth > 0:
        pygame.draw.rect(self.screen, borderColor, rect)

    innerRect = pygame.Rect(
        x + borderWidth,
        y + borderWidth,
        width - 2 * borderWidth,
        height - 2 * borderWidth
    )
    pygame.draw.rect(self.screen, buttonColor, innerRect)

    return innerRect

def drawTextOnRect(self, text, buttonRect, textColor, font):
    textSurface = font.render(text, True, textColor)
    textRect = textSurface.get_rect(center=buttonRect.center)
    self.screen.blit(textSurface, textRect)

    return

def drawText(self, text, point, font, color, lineSpacing=5):
    lines = text.splitlines()
    x, y = point
    for line in lines:
        textSurface = font.render(line, True, color)
        self.screen.blit(textSurface, (x, y))
        y += textSurface.get_height() + lineSpacing

    return

def drawButton(self, button):

    borderColor = [255, 222, 89] if button["selected"] else button["borderColor"]
    backgroundColor = [255, 222, 89] if button["selected"] else button["backgroundColor"]

    font = self.FONT if button["fontSize"] == "big" else self.SMALL_FONT

    rect = self.drawRect(self.getY(button["y"]), self.BUTTON_WIDTH, self.BUTTON_HEIGHT, backgroundColor,
         borderColor, button["borderWidth"])
    self.drawTextOnRect(button["text"], rect, button["textColor"], font)

    return

def drawMainMenu(self):

    self.screen.fill(self.DARK_PURPLE)
    self.screen.blit(self.imgBackground, (0, 0))
    for button in self.mainMenuButtons.values():
        self.drawButton(button)

    pygame.display.flip()
    return

def drawControlsMenuMenu(self):
```

```
        self.screen.fill(self.DARK_PURPLE)
        self.screen.blit(self.imgBackground, (0, 0))
        for button in self.controlsMenuButtons.values():
            self.drawButton(button)

        joyconImg = pygame.image.load("../imgs/joy-con.png")
        joyconImg = pygame.transform.scale(joyconImg, (self.WIDTH*0.7, self.HEIGHT*0.7))

        imgWidth, imgHeight = joyconImg.get_size()
        x = (self.WIDTH - imgWidth) // 2
        y = (self.HEIGHT - imgHeight) // 2

        self.screen.blit(joyconImg, (x, y+100))

        pygame.display.flip()
        return

    def drawBluetoothPairInstructions(self, option):
        self.screen.fill(self.DARK_PURPLE)
        self.screen.blit(self.imgBackground, (0, 0))
        header = {
            "y": 70,
            "backgroundColor": self.LIGHT_PURPLE,
            "borderColor": self.LIGHT_PURPLE,
            "borderWidth": 10,
            "text": "INSTRUCCIONES␣DE␣EMPAREJAMIENTO␣DE␣LOS␣JOY-CON",
            "fontSize": "big",
            "textColor": self.DARK_PURPLE,
            "selected": 1
        }

        #TODO: Mejorar las instrucciones
        instructions = """
        Manten presionado el pequeo botn negro de emparejamiento ubicado al costado del riel de cada Joy-Con
        hasta que las luces comiencen a destellar.

        En cuanto escuches los destellos:

        - PRESIONA ZL y ZR al mismo tiempo para usarlo como un nico control.
        - PRESIONA SL y SR en cada joycon para asignarlos como controles individuales.
        """
        self.drawButton(header)
        self.drawText(instructions, (150,150), self.SMALL_FONT, self.LIGHT_PURPLE)

        imgPath = "../imgs/"
        imgPath += "pair-joycons.png" if option == "pairing" else "joy-con.png"
        joyconImg = pygame.image.load(imgPath)
        joyconImg= pygame.transform.scale(joyconImg, (self.WIDTH*0.6, self.HEIGHT*0.7))

        imgWidth, imgHeight = joyconImg.get_size()
        x = (self.WIDTH - imgWidth) // 2
        y = (self.HEIGHT - imgHeight) // 2

        self.screen.blit(joyconImg, (x, y+100))


        pygame.display.flip()
        return

    def drawNewGames(self, games):
        self.screen.fill(self.DARK_PURPLE)
        self.screen.blit(self.imgBackground, (0, 0))
        header = {
            "y": 70,
            "backgroundColor": self.LIGHT_PURPLE,
            "borderColor": self.LIGHT_PURPLE,
            "borderWidth": 10,
            "text": "NUEVOS␣JUEGOS",
            "fontSize": "big",
            "textColor": self.DARK_PURPLE,
            "selected": 1
        }
```

```
        text = ""

        if len(games) == 0:
            text = "No se agregaron nuevos juegos :c"
        else:
            for game in games:
                text += game + "\n"

        self.drawButton(header)
        self.drawText(text, (150,150), self.SMALL_FONT, self.LIGHT_PURPLE)

        pygame.display.flip()
        return

    def setConsole(self, console):
        self.mainMenuButtons["consoleOptions"]["text"] = console
        return

    def setGame(self, game):
        self.mainMenuButtons["gameOptions"]["text"] = game
        return

    def setMenuOption(self, option):
        for menuOption in self.mainMenuButtons.keys():
            self.mainMenuButtons[menuOption]["selected"] = 0

        menuOptions = list(self.mainMenuButtons.keys())[:4]
        self.mainMenuButtons[menuOptions[option]]["selected"] = 1
        return

    def removeDisplay(self):
        pygame.display.quit()

    def hideDisplay(self):
        os.environ["SDL_VIDEODRIVER"] = "dummy"
        pygame.display.init()

    def showDisplay(self):
#       pygame.quit()
        time.sleep(2)
        os.environ["SDL_VIDEODRIVER"] = "kmsdrm"
#       pygame.init()
        pygame.display.init()
        self.screen = pygame.display.set_mode((self.WIDTH, self.HEIGHT))
```

—

### 5.4.6 Arcadiax

Por último tenemos la clase que integra todo lo que ha sido descrito con anterioridad. Esto nos permite conectar los controles Joy-Con, visualizar los juegos desde el menú principal y realizar la detección de USB para la incorporación de nuevas ROMs, así como el uso del emulador.

Para manejar los Joy-Cons, en la función `connectJoyCons` se establece un hilo que intenta conectar los dispositivos de manera continua, mientras que la interfaz gráfica muestra instrucciones para guiar al usuario. Esto permite que los controles estén listos antes de interactuar con el menú o los juegos. Así mismo, hay métodos para modificar el menú, de acuerdo a la selección que haya realizado el usuario interpretando la entrada del joystick.

Por otro lado, se utiliza otro hilo para monitorear la conexión de dispositivos USB pata detectar los nuevos juegos mientras se encuentre ejecutándose el programa. El código completo de la clase se puede ver a continuación.

```
import Interface
import JoyCons
import ROMSManager
import USBManager
import threading
import time
import pygame
import subprocess
import os
import signal
import uinput
```

```
class Arcadiax:
    def __init__(self):
        self.interface = Interface.Interface()
        self.joycons = JoyCons.JoyCons()
        self.usbDetection = USBManager.USBManager()
        self.play = True
        self.newGames = False
        self.newUSBGames = []
        self.gamesDetectionFlag = True
        self.gamesDetectionThread = None
        self.mainMenuOptions = {
            "menu" : 0,
            "console" : 0,
            "game" : 0,
            "play" : 0,
        }
        self.romsManager = ROMSManager.ROMSManager()
        self.roms = self.romsManager.getROMSGroupByConsole()
        self.mednafen = None

        pygame.init()
        return

    def start(self):
        self.joycons.initJoyCon1()

        self.gamesDetectionThread = threading.Thread(target=self.gamesDetection)
        self.gamesDetectionThread.start()

        while self.play:

            pygame.event.pump()

            for event in pygame.event.get():
                if event.type == pygame.JOYBUTTONDOWN:
                    print(event.button)
                if event.type == pygame.KEYDOWN:
                    if event.key == pygame.K_e:
                        self.exit()

            if self.mainMenuOptions["play"]:
                if self.joycons.joycon1.get_button(11): # home Button
                    os.killpg(os.getpgid(self.mednafen.pid), signal.SIGTERM)
                    self.mainMenuOptions["play"] = 0
                    self.interface.removeDisplay()
                    self.interface.showDisplay()

                if self.joycons.joycon1.get_button(4): # Screenshot button
                    print("hola hola")
                    events = [uinput.KEY_LEFTALT, uinput.KEY_LEFTSHIFT, uinput.KEY_1]
                    with uinput.Device(events) as device:
                        time.sleep(1)
                        device.emit_combo([uinput.KEY_LEFTALT, uinput.KEY_LEFTSHIFT, uinput.KEY_1])

            if not self.mainMenuOptions["play"]:
                joystickX = self.joycons.joycon1.get_axis(0)
                joystickY = self.joycons.joycon1.get_axis(1)
                self.changeMainMenuOptions(joystickX, joystickY)
                self.updateMainMenu()

                if self.newGames:

                    self.interface.drawNewGames(self.newUSBGames)
                    time.sleep(10)

                    self.newGames = False
                    self.newUSBGames = []

                self.interface.drawMainMenu()

                if self.joycons.joycon1.get_button(1): # A Button
                    self.selectMainMenuOptions()
```

```
            pygame.time.wait(100)

        return

    def connectJoyCons(self):

        connectJoyConsThread = threading.Thread(target=self.joycons.connectJoyCons)
        connectJoyConsThread.start()

        while connectJoyConsThread.is_alive():
            self.interface.drawBluetoothPairInstructions("pairing")
            time.sleep(10)
            self.interface.drawBluetoothPairInstructions("connect")
            time.sleep(10)

        self.interface.drawBluetoothPairInstructions("connect")
        time.sleep(10)

    def changeMainMenuOptions(self, joystickX, joystickY):
        if joystickY < -0.2: #UP
            self.mainMenuOptions["menu"] -= 1
            if self.mainMenuOptions["menu"] < 0:
                self.mainMenuOptions["menu"] = 3

        elif joystickY > 0.2: #DOWN
            self.mainMenuOptions["menu"] += 1
            if self.mainMenuOptions["menu"] > 3:
                self.mainMenuOptions["menu"] = 0

        elif joystickX < -0.2: #LEFT
            if self.mainMenuOptions["menu"] == 0:
                self.mainMenuOptions["console"] -= 1
                if self.mainMenuOptions["console"] < 0:
                    self.mainMenuOptions["console"] = len(self.roms) - 1
                self.mainMenuOptions["game"] = 0

            elif self.mainMenuOptions["menu"] == 1:
                self.mainMenuOptions["game"] -= 1
                if self.mainMenuOptions["game"] < 0:
                    console = list(self.roms.keys())
                    console = console[self.mainMenuOptions["console"]]
                    self.mainMenuOptions["game"] = len(self.roms[console]) - 1

        elif joystickX > 0.2: #RIGHT
            if self.mainMenuOptions["menu"] == 0:
                self.mainMenuOptions["console"] += 1
                if self.mainMenuOptions["console"] > len(self.roms) - 1:
                    self.mainMenuOptions["console"] = 0
                self.mainMenuOptions["game"] = 0

            elif self.mainMenuOptions["menu"] == 1:
                self.mainMenuOptions["game"] += 1
                console = list(self.roms.keys())
                console = console[self.mainMenuOptions["console"]]
                if self.mainMenuOptions["game"] > len(self.roms[console]) - 1:
                    self.mainMenuOptions["game"] = 0

    def updateMainMenu(self):
        self.interface.setMenuOption(self.mainMenuOptions["menu"])
        console = list(self.roms.keys())
        console = console[self.mainMenuOptions["console"]]
        self.interface.setConsole(console)
        self.interface.setGame(self.roms[console][self.mainMenuOptions["game"]])
        return

    def selectMainMenuOptions(self):
        if self.mainMenuOptions["menu"] == 2: #PLAY
            self.interface.removeDisplay()
            self.mainMenuOptions["play"] = 1
            console = list(self.roms.keys())
            console = console[self.mainMenuOptions["console"]]
            game = self.roms[console][self.mainMenuOptions["game"]]
```

```
            self.mednafen = subprocess.Popen(["/usr/games/mednafen", "/home/emilia/arcadiax/roms/"+game],
                 preexec_fn=os.setsid)
            self.interface.hideDisplay()
        elif self.mainMenuOptions["menu"] == 3: #CONTROLS
            print("modificando el ajsute de los controles")
            self.powerOff()
        return

    def gamesDetection(self):
        while self.gamesDetectionFlag:
            mountPoint = self.usbDetection.usbDetection()

            if mountPoint == None:
                break

            games = self.romsManager.getROMSFromADir(mountPoint)
            for game in games:
                gameAux = self.romsManager.changeROMName(game)
                if not self.romsManager.ROMExist(gameAux):
                    self.romsManager.copyROM(mountPoint, game)
                    self.newUSBGames.append(game)

            self.newGames = True
            self.roms = ROMSManager.ROMSManager().getROMSGroupByConsole()
            self.usbDetection.flag = True
        return

    def exit(self):
        self.play = False
        self.usbDetection.flag = False
        self.gamesDetectionFlag = False
        self.gamesDetectionThread.join()
        self.joycons.disconnectJoyCons()
        return

    def powerOff(self):
        self.exit()
        subprocess.run(['sudo', 'shutdown', 'now'])
```

Y para finalizar es necesario crear un programa principal que mande a llamar a la clase anterior `main.py`. El cual, al ejecutarse nos debería permitir vincular nuestros controles después de presionar los botones de emparejamiento que se encuentran en el centro de los rieles de estos. Posteriormente, tenemos que presionar los gatillos (ZL y ZR) para que `joycond` los configure como si fueran uno solo, y en seguida visualizar el menú principal donde podremos seleccionar la consola y el juego que queremos emular.

```
import Arcadiax
import pygame
arcAux = Arcadiax.Arcadiax()
arcAux.connectJoyCons()
pygame.quit()
arcadiax = Arcadiax.Arcadiax()
arcadiax.start()
```

## 5.5 Configuración para que el programa se ejecute al iniciar el sistema

Este es el último paso para transformar la Raspberry Pi en una consola de videojuegos. Para conseguirlo es necesario editar el archivo `~/.bashrc`. Este se ejecuta cuando un usuario inicia sesión. Hasta el final, debemos colocar la instrucción que ejecute el programa `main.py`. Para ello, tenemos que cambiarnos al directorio donde se encuentra nuestro programa. El final del contenido del archivo debe verse de la siguiente manera:

```
cd arcadiax\modules
python3 main.py
```

Y al reiniciar el sistema, si todo ha sido configurado de manera adecuada, tendremos una consola de videojuegos. Si por algún motivo es necesario realizar más cambios, mientras nos encontramos en el menú de selección de videojuegos, si precionamos la tecla e, el programa terminará y podremos usar la consola.

El video demostrativo se puede visualizar con el siguiente enlace a youtube: https://youtu.be/q7lK8L$_D$lJs

## Cuestionario

- En caso de querer conectar otro gamepad a través de Bluetooth, ¿qué modificaciones al código serían necesarias?

  Los ajustes requeridos incluirían modificaciones en el archivo `bluetooth-get-mac-joycons.sh`, donde sería necesario actualizar la lógica para identificar el nuevo dispositivo y devolver únicamente su dirección MAC correspondiente. Adicionalmente, en el módulo `Arcadiax`, se debería analizar y ajustar el mapeo de los botones del nuevo control, adaptando las configuraciones relacionadas con la selección de opciones en el menú, el botón designado para remapear los controles y el comando para salir del juego.

- ¿Por qué no es la mejor solución crear un demonio para que el sistema se ejecute inmediatamente al finalizar el proceso de arranque?

  Esto se debe a que algunos servicios, especialmente los relacionados con Bluetooth, podrían no estar completamente disponibles en ese momento, lo que podría generar retrasos o fallos en el funcionamiento esperado.

- ¿Es posible integrar otro emulador que soporte juegos que puedan interactuar con los sensores de giroscopio de los Joy-Con?7

  Es posible integrar otro emulador que soporte otras consolas. Para ello, sería necesario modificar el módulo `Arcadiax` para que pueda identificar qué consola se está emulando y asociarla con la aplicación correspondiente. En el caso de emuladores como Cemu (para Wii U) o Dolphin (para Wii y GameCube), que soportan juegos con sensores de movimiento, se debe verificar tanto la compatibilidad como los requisitos de hardware y software para asegurar su correcto funcionamiento. Además, será necesario instalar y configurar los drivers adecuados que permitan la comunicación entre los sensores de giroscopio de los Joy-Con y el emulador.

## Conclusiones

Se ha desarrollado una consola de videojuegos portátil que aprovecha las capacidades de la Raspberry Pi 4, como lo es la conectividad Bluetooth, que permite la vinculación inalámbrica de controles Joy-Con, ofreciendo una experiencia más cómoda y práctica para el usuario.

Sin embargo, es importante considerar las limitaciones del hardware del dispositivo. Aunque es técnicamente posible integrar más emuladores o funcionalidades, los recursos disponibles, como la capacidad de procesamiento, memoria y almacenamiento, son limitados. Esto podría impactar en el rendimiento al ejecutar aplicaciones o juegos que demanden mayor potencia.

En conclusión, este proyecto muestra cómo es posible construir una consola portátil funcional. Sin embargo, existen áreas para futuras mejoras, como el diseño de una interfaz más intuitiva y la optimización del proceso de conexión de los controles. Actualmente, si este proceso de conexión no se realiza de manera específica, el programa puede fallar.

## Referencias


[1] Arch Linux. *Getty*. 2024. URL: https://wiki.archlinux.org/title/Getty.

[2] Arch Linux. *Silent boot*. 2024. URL: https://wiki.archlinux.org/title/Silent_boot.

[3] *Capítulo 3. Gestión de servicios con systemd*. 2024. URL: https://docs.redhat.com/es/documentation/red_hat_enterprise_linux/8/html/configuring_basic_system_settings/index.

[4] Debian. *Plymouth*. 2024. URL: https://wiki.debian.org/es/plymouth.

[5] Debian. *systemd Services*. 2024. URL: https://wiki.debian.org/systemd/Services.

[6] *Emuladores*. 2024. URL: https://www.ibm.com/docs/es/aix/7.2?topic=concepts-emulators.

[7] *Mednafen*. 2024. URL: https://mednafen.github.io/.

[8] *Raspberry Pi 4 Tech Specs*. 2024. URL: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/.

[9] Red Hat. *10.3. Plymouth*. 2024. URL: https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/7/html/desktop_migration_and_administration_guide/plymouth.

[10] *The Standard for Embedded Accelerated 3D Graphics*. URL: https://www.khronos.org/opengles/.

[11] Ubuntu. *Change the splash screen*. 2024. URL: https://ubuntu.com/core/docs/splash-screen.