Proyecto Final:
Consola de videojuegos Retro

Fundamentos de Sistemas Embebidos
Luis Fernando González Zambrano
Mario Abraham López Diego

1. Objetivo

El alumno implementará un centro de entretenimiento emulación de una o más consolas de videojuegos usando Mednafen.

2. Introducción

Este proyecto busca desarrollar un centro de entretenimiento retro, combinando hardware y software para emular consolas clásicas de videojuegos. Utilizando una Raspberry Pi 5 como tarjeta controladora y un gamepad basado en RP2040, se busca permitir al usuario jugar títulos de consolas como NES, SNES y GameBoy Advance, esto mediante el emulador Mednafen.

El objetivo principal es desarrollar un sistema que arranque directamente al emulador con una animación personalizada y que permita controlar todos los aspectos mediante un gamepad, excluyendo el uso de teclado o mouse. Además, se integrará una galería inicial de juegos retro y un mecanismo para agregar nuevos juegos automáticamente al insertar una memoria USB, lo que proporcionará al usuario una experiencia amigable y sencilla de usar.

3. Antecedentes

Para realizar este proyecto, es necesario tener los conocimientos necesarios sobre los siguientes componentes.

3.1. Raspberry Pi

Una Raspberry Pi es una computadora compacta y de bajo costo desarrollada por la Fundación Raspberry Pi. Está equipada con un procesador ARM, memoria RAM que varía según el modelo (desde 512MB hasta 8GB), y utiliza tarjetas microSD para el almacenamiento del sistema operativo y los archivos. Además, cuenta con diversos puertos de conectividad como USB, HDMI, Ethernet, y opciones integradas de Wi-Fi y Bluetooth, lo que la hace extremadamente versátil para una variedad de aplicaciones. [1]

El uso de una Raspberry Pi es bastante sencillo y accesible. Primero, se descarga e instala un sistema operativo, como Raspberry Pi OS, en una tarjeta microSD. Luego, la tarjeta se inserta en la Raspberry Pi y se conecta a un monitor, teclado, ratón y fuente de alimentación. Una vez encendida,

se sigue el proceso de configuración inicial. Después de esto, la Raspberry Pi se puede utilizar como una computadora convencional para navegar por internet, editar documentos, aprender a programar o ejecutar proyectos de electrónica.

Sus pines GPIO permiten integrar dispositivos externos como sensores, motores y luces, lo que la hace ideal para proyectos de robótica, automatización y sistemas embebidos. Su uso es tan flexible que puede funcionar como una computadora convencional o como un componente clave en proyectos personalizados. [2]

3.2. Raspberry Pi Pico

La Raspberry Pi Pico es un microcontrolador diseñado para manejar tareas específicas de control y procesamiento en proyectos electrónicos y de automatización. A diferencia de las computadoras tradicionales Raspberry Pi, la Pico no ejecuta sistemas operativos completos, sino que se programa para realizar tareas en tiempo real, como leer sensores, controlar motores o manejar señales digitales. Gracias a su diseño basado en el microcontrolador RP2040, es ideal para aplicaciones que requieren bajo consumo energético, tamaño compacto y capacidad para interactuar directamente con hardware externo.

Se utiliza principalmente en proyectos de Internet de las Cosas (IoT), sistemas embebidos, robótica, automatización industrial y dispositivos portátiles. Entre sus funciones destacan la adquisición de datos de sensores, generación de señales PWM para control de motores o iluminación, procesamiento de datos en tiempo real y comunicación con otros dispositivos mediante interfaces como I2C, SPI o UART. También es ampliamente utilizada en entornos educativos para enseñar conceptos de programación y electrónica debido a su facilidad de uso y soporte para lenguajes accesibles como MicroPython y CircuitPython. [3, 4]

Sus principales características abarcan:

- Procesador potente: La Raspberry Pi Pico cuenta con un microcontrolador RP2040 de doble núcleo basado en ARM Cortex-M0+ a 133 MHz, ofreciendo un excelente equilibrio entre potencia y eficiencia energética.
- Memoria integrada: Incluye 264 KB de SRAM y 2 MB de memoria flash QSPI, lo que permite manejar aplicaciones complejas y almacenar programas directamente en el dispositivo.
- Compatibilidad con múltiples interfaces: Ofrece una amplia variedad de pines GPIO configurables y soporte para protocolos comunes como I2C, SPI, UART, PWM y ADC, ideal para conectar sensores y módulos periféricos.
- Alta flexibilidad en programación: Es compatible con lenguajes de programación como MicroPython, CircuitPython y C/C++, permitiendo a los usuarios elegir la opción que mejor se ajuste a su proyecto o nivel de experiencia. [3, 4]

3.3. CircuitPython

CircuitPython es una versión simplificada de Python 3 diseñada para programar microcontroladores, creada por Adafruit. Su principal objetivo es facilitar el desarrollo de proyectos electrónicos, permitiendo a los usuarios escribir código directamente en un microcontrolador que aparece como una unidad USB al conectarlo a la computadora. Es especialmente útil para principiantes, ya que elimina la necesidad de herramientas de desarrollo complicadas y proporciona una forma intuitiva de interactuar con el hardware.

A diferencia de otras plataformas de programación para microcontroladores, CircuitPython enfatiza la simplicidad y la accesibilidad. Permite modificar el código en tiempo real sin necesidad

de reiniciar el dispositivo, lo que acelera el proceso de prueba y desarrollo. También cuenta con una amplia colección de bibliotecas que ofrecen soporte para una variedad de sensores, motores y pantallas, haciéndolo ideal para proyectos de prototipado rápido o educativos. [5, 6]

Principales características:

- Rápida configuración del entorno: CircuitPython permite iniciar proyectos de manera inmediata. Basta con crear un archivo, escribir el código, guardarlo y se ejecutará automáticamente en el microcontrolador, sin necesidad de procesos de compilación ni transferencias manuales al dispositivo.
- Actualización dinámica del código: Al almacenar el código directamente en el sistema de archivos del dispositivo, puedes editarlo en cualquier momento sin interrumpir el flujo de trabajo. Además, puedes manejar múltiples archivos para experimentar con diferentes configuraciones de manera eficiente.
- Soporte de almacenamiento interno: La unidad de almacenamiento del microcontrolador puede usarse para registrar datos, reproducir archivos de audio o interactuar con otros tipos de archivos directamente desde el dispositivo.
- Amplias capacidades de hardware: Incluye una vasta colección de bibliotecas y controladores que facilitan la interacción con sensores, módulos especializados y componentes externos, optimizando el control de hardware desde el microcontrolador. [6]

4. Material

Es necesario contar con una Raspberry Pi con sistema operativo Raspbian e interprete de Python instalado. Además, es necesario contar con:

- 1 Tarjeta microcontroladora Raspberry Pi Pico o UNIT DualMCU RP2040 + ESP32
- Una tarjeta de memoria microSD de al menos 4 GB (se recomiendan 8GB)
- Un monitor con soporte para HDMI o cable convertidor de HDMI a VGA/DVI
- Cable adaptador micro HDMI a HDMI (Raspberry Pi 4 y 5)
- Un teclado USB
- Un mouse USB
- Una fuente de alimentación de 5V@2A con adaptador microUSB (o adaptador USB-C para Raspberry Pi 4 y 5)
- 1 cable USB-C con soporte para datos
- Cables, botones y conectores varios

5. Configuración de la Raspberry Pi

5.1. Instalación de dependencias

Primero se actualiza los paquetes existentes y el sistema operativo:

```
sudo apt-get update
sudo apt-get upgrade
```

Luego se instalan los paquetes y servicios necesarios para ejecutar la aplicación. Xorg y Openbox proporcionan las herramientas para ejecutar el servidor gráfico. Python3 se usará para ejecutar (`game_gallery.py`), mientras que Mednafen es el emulador para jugar ROMs retro. Los paquetes adicionales de Python permiten manejar interfaces gráficas, imágenes y eventos de hardware.

```
sudo apt-get install xorg
sudo apt-get install openbox
sudo apt-get install python3 python3-pip mednafen
sudo apt-get install python3-tk python3-pil python3-pil.imagetk python3-pyudev
```

5.2. Configuración de inicio de sesión automático

Posteriormente se configura la Raspberry Pi para que inicie sesión automáticamente en la consola principal (tty1). Se crea el directorio para anular la configuración predeterminada y se edita el archivo `override.conf`:

```
sudo mkdir -p /etc/systemd/system/getty@tty1.service.d
sudo nano /etc/systemd/system/getty@tty1.service.d/override.conf
    [Service]
    ExecStart=
    ExecStart=-/sbin/agetty --autologin admin --noclear %I $TERM
```

5.3. Configuración de arranque automático

Se configura Openbox para iniciar el programa, para lo que se crea el directorio de configuración de Openbox y se edita el archivo `autostart` para iniciar el programa automáticamente:

```
mkdir -p ~/.config/openbox
nano ~/.config/openbox/autostart
    python3 /home/admin/proyecto_final/game_gallery.py
```

Luego se configura el arranque con `startx`. Se edita `/.bashrc` para iniciar el servidor gráfico con startx automáticamente:

```
nano ~/.bashrc
    startx -- -nocursor >/dev/null 2>&1
```

Posteriormente, se edita el archivo `cmdline.txt` para redirigir los mensajes a otra consola y limpiar el arranque:

```
sudo nano /boot/firmware/cmdline.txt
    ... ... quiet loglevel=3 console=tty3
```

5.4. Configuración de logo al iniciar el sistema

Para mostrar un logo personalizado en el arranque del sistema se instala `fbi`, una herramienta para mostrar imágenes en el framebuffer, se copia la imagen del logotipo al directorio `/etc/` y se crea un servicio de `systemd` para mostrar el logotipo al inicio:

```
sudo apt-get install fbi
sudo cp /home/admin/Games/images/logo.png /etc/splash.png
sudo nano /etc/systemd/system/splashscreen.service
    [Unit]
    Description=Mostrar imagen de inicio personalizada
    DefaultDependencies=no
    After=local-fs.target
    [Service]
    ExecStart=/usr/bin/fbi -d /dev/fb0 --noverbose -a /etc/splash.png
    StandardInput=tty
```

```
StandardOutput=tty
[Install]
WantedBy=sysinit.target
```

Finalmente se habilita y prueba el servicio:

```
sudo systemctl daemon-reload
sudo systemctl enable splashscreen
sudo systemctl start splashscreen
```

5.5. Manejo de dispositivos de almacenamiento externo

Al trabajar con un entorno de Raspberry Pi OS Lite, el sistema no monta automáticamente los dispositivos USB conectados, por lo que fue necesario instalar `usdisks2` y establecer el servicio para que inicie automáticamente con el sistema:

```
sudo apt-get install udisks2
sudo systemctl enable udisks2
```

Posteriormente, creamos una nueva regla para manejar el montaje automático de dispositivos USB:

```
sudo nano /etc/udev/rules.d/99-udisks2-auto-mount.rules

ACTION=="add", SUBSYSTEM=="block", ENV{ID_FS_USAGE}=="filesystem", RUN+="/usr/bin
~~~/udisksctl-mount--b-$env{DEVNAME}"
ACTION=="remove", SUBSYSTEM=="block", ENV{ID_FS_USAGE}=="filesystem", RUN+="/usr/bin
~~~/udisksctl-unmount--b-$env{DEVNAME}"
```

Finalmente recargamos la reglas de `udev`:

```
sudo udevadm control --reload-rules
sudo udevadm trigger
```

6. Códigos - Software

Este proyecto se divide en la parte de hardware y la parte de software. Esta última es la que cuenta con el código más extenso y que incluye una mayor variedad de funciones, las cuales se describirán a continuación. El programa `game_gallery.py` es el que se ejecutará automáticamente al encender la Raspberry Pi y es el que contiene todas las funciones necesarias en un único archivo.

Primeramente se hace la importación de las librerías necesarias:

```
import os
import subprocess
import tkinter as tk
from tkinter import messagebox
from tkinter import ttk
from PIL import Image, ImageTk
import pyudev
import threading
import time
import shutil
```

Maneja operaciones del sistema operativo
Ejecuta comandos del sistema
Librería para crear interfaces gráficas de usuario
Para mostrar diálogos emergentes en la interfaz
Widgets adicionales para Tkinter
Librería para manejar imágenes
Para monitorizar eventos de dispositivos USB
Para ejecutar ciertas funciones en hilos separados
Manejo del tiempo, como sleep
Operaciones relacionadas con el sistema de archivos

6.1. Clase `GameCarouselApp`

En esta parte se inicializa la clase principal de la aplicación, se definen atributos y métodos como los directorios del sistema, puntos de montaje de usb, el proceso del juego actual, la ventana principal del programa y sus diferentes atributos, se obtiene la lista de juegos disponibles, se crea los elementos de la interfaz gráfica, se asignan eventos a teclas específicas para navegar en la galería y se actualiza el carrusel de juegos y la barra de progreso.

```
class GameCarouselApp:
    def __init__(self, master, roms_directory, images_directory, logo_path):
        self.master = master
        self.roms_directory = roms_directory
        self.images_directory = images_directory
        self.logo_path = logo_path
        self.usb_mount_point = "/media/usb"
        self.current_game_process = None
        self.master.title("Carrusel de Juegos")
        self.master.attributes("-fullscreen", True)
        self.start_usb_monitoring()
        self.roms = self.get_roms()
        self.current_index = 0
        self.create_widgets()
        self.master.bind("A", self.move_left)
        self.master.bind("D", self.move_right)
        self.master.bind("<Return>", self.play_game)
        self.master.bind("<Escape>", self.return_to_gallery)
        self.update_carousel()
        self.update_progress_bar()
```

6.2. Creación de la interfaz gráfica

La función `create_widgets` es responsable de construir la interfaz gráfica de la aplicación. Esta crea diferentes elementos visuales, incluyendo un marco principal, un encabezado con un logo, un área para mostrar los juegos en un carrusel horizontal y una barra de progreso. Cada widget se configura y coloca en su correspondiente marco usando el gestor de geometría de `Tkinter`. El logo se sitúa en la parte superior, mientras que los juegos se muestran en el centro, con su título sobre la imagen de portada. La barra de progreso se encuentra en la parte inferior, indicando la posición del juego seleccionado.

```
def create_widgets(self):
    self.main_frame = tk.Frame(self.master, bg="#1E1940")
    self.main_frame.pack(fill=tk.BOTH, expand=True)
    self.header_frame = tk.Frame(self.main_frame, bg="#1E1940")
    self.header_frame.pack(side=tk.TOP, fill=tk.X, pady=10, padx=10)
    self.logo_label = tk.Label(self.header_frame, bg="#1E1940")
    self.logo_label.pack(side=tk.LEFT)
    if self.logo_path and os.path.exists(self.logo_path):
        [CONFIGURACIONES VARIAS]
        self.logo_label.image = tk_logo_img
    self.message_label = tk.Label(self.main_frame, text="Bienvenido
------Selecciona un juego:\n", bg="#1E1940", fg="#F2AEE0")
    self.message_label.pack(side=tk.TOP, pady=(10, 0))
    self.carousel_frame = tk.Frame(self.main_frame, bg="#1E1940")
    self.carousel_frame.pack(fill=tk.BOTH, expand=True)
    self.game_labels = {
        [CONFIGURACIONES VARIAS]
```

```
    }
    self.game_labels["left"].pack(side=tk.LEFT, expand=True, padx=10)
    self.game_labels["center"].pack(side=tk.LEFT, expand=True, padx=10)
    self.game_labels["right"].pack(side=tk.LEFT, expand=True, padx=10)
    self.progress_frame = tk.Frame(self.main_frame, bg="#1E1940")
    self.progress_frame.pack(side=tk.BOTTOM, fill=tk.X, pady=10)
    self.progress_bar = ttk.Progressbar(self.progress_frame, )
    self.progress_bar.pack(fill=tk.X, padx=20)
```

En la siguiente función se obtiene y ordena alfabéticamente los juegos disponibles en el directorio:

```
def get_roms(self):
    return sorted([
        rom for rom in os.listdir(self.roms_directory)
        if rom.endswith('.gba') or rom.endswith('.sfc')
    ])
```

También se buscan las portadas de los juegos en el directorio de imágenes. La imagen debe tener el formato adecuado y estar nombrada de la misma manera que la ROM asociada:

```
def get_image_path(self, rom_name):
    base_name = os.path.splitext(rom_name)[0]
    for ext in ['.png', '.jpg', '.jpeg']:
        image_path = os.path.join(self.images_directory, f"{base_name}{ext}")
        if os.path.exists(image_path):
            return image_path
```

6.3. Monitoreo y manejo de dispositivos USB

Inicia el monitoreo de inserciones de USB en un hilo separado:

```
def start_usb_monitoring(self):
    self.usb_thread = threading.Thread(target=self.monitor_usb, daemon=True)
    self.usb_thread.start()

def monitor_usb(self):
    context = pyudev.Context()
    monitor = pyudev.Monitor.from_netlink(context)
    monitor.filter_by(subsystem='block', device_type='disk')
    observer = pyudev.MonitorObserver(monitor, self.handle_device_event)
    observer.start()
```

Luego se maneja el evento de inserción de un dispositivo USB mediante la creación de un hilo secundario para llevar a cabo el proceso completo:

```
def handle_device_event(self, action, device):
    if action == "add" and device.get("ID_BUS") == "usb":
        mount_point = self.get_mount_point(device)
        if mount_point:
            threading.Thread(target=self.handle_usb_inserted,
                args=(mount_point,), daemon=True).start()
```

Después, se obtiene el punto de montaje del dispositivo USB:

```
def get_mount_point(self, device):
    device_name = device.device_node
    try:
        mount_output = subprocess.check_output(f"lsblk -o MOUNTPOINT -nr
{device_name}", shell=True).decode().strip()
```

```
        return mount_output if mount_output else None
    except subprocess.CalledProcessError:
        return None
```

La siguiente función maneja la inserción del USB y copia nuevos juegos al directorio. Si hay un juego en ejecución, lo cierra y continúa con el proceso. Busca en el directorio raiz del dispositivo archivos con las extensiones .gba, .sfc y .smc, si encuentra, valida que los juegos no estén ya en la galería de juegos actual y si hay nuevos juegos, los copia al directorio de juegos. Para cada caso muestra un mensaje al usuario con el resultado del proceso. Finalmente desmonta el dispositivo del sistema para poder extraerlo de manera segura y actualiza la galería de juegos.

```
def handle_usb_inserted(self, mount_point):
    if self.current_game_process and self.current_game_process.poll() is None:
        self.current_game_process.terminate()
        self.current_game_process.wait()
    valid_extensions = (".gba", ".sfc", ".smc")
    new_games = [
        file for file in os.listdir(mount_point)
        if file.endswith(valid_extensions) and not os.path.exists(os.path.join
            (self.roms_directory, file))
    ]
    if not new_games:
        self.show_notification("No se encontraron nuevos juegos en la USB.",
            duration=5000)
        subprocess.run(["sudo", "umount", mount_point])
        self.start_usb_monitoring()
        return
    self.show_notification("Copiando juegos desde la USB...")
    for game in new_games:
        source_path = os.path.join(mount_point, game)
        destination_path = os.path.join(self.roms_directory, game)
        try:
            shutil.copy2(source_path, destination_path)
        except Exception as e:
            print(f"Error al copiar {game}: {e}")
    self.show_notification(f"Se copiaron {len(new_games)} nuevo(s) juego(s)
------desde la USB. Ahora puedes retirar la USB de forma segura.", duration=5000)
    subprocess.run(["sudo", "umount", mount_point])
    self.roms.extend(new_games)
    self.roms = sorted(self.roms)
    self.update_carousel()
    self.start_usb_monitoring()
```

6.4. Actualización del carrusel y la ventana de juegos

Cuando ocurre el proceso de copiado de juegos mediante una usb se actualizan las vistas con las siguientes funciones. La primera función actualiza la visualización del carrusel con el juego seleccionado y los vecinos:

```
def update_carousel(self):
    left_index = (self.current_index - 1) % len(self.roms)
    right_index = (self.current_index + 1) % len(self.roms)
    self.update_game_window(self.game_labels["left"], self.roms[left_index], ...)
    self.update_game_window(self.game_labels["center"], self.roms[self.current_index]
        , ...)
    self.update_game_window(self.game_labels["right"], self.roms[right_index], ...)
```

La siguiente función actualiza la información visual de cada juego (imagen, título):

```
def update_game_window(self, label, rom_name, size, is_selected=False):
    image_path = self.get_image_path(rom_name)
    if not image_path:
        image_path = self.get_image_path("logo")
    img = Image.open(image_path).resize(size, Image.ANTIALIAS)
    tk_img = ImageTk.PhotoImage(img)
    label.config(image=tk_img)
    label.image = tk_img
    title = os.path.splitext(rom_name)[0]
    if len(title) > 23:
        title = title[:20] + "..."
    label.config([CONFIGURACIONES VARIAS])
```

Finalmente, se actualiza la barra de progreso según la posición actual en el carrusel:

```
def update_progress_bar(self):
    total_games = len(self.roms)
    self.progress_bar["maximum"] = total_games
    self.progress_bar["value"] = self.current_index + 1
```

6.5. Ejecución del juego seleccionado

La función `play_game` es responsable de iniciar el juego seleccionado desde el carrusel. Primero, obtiene el nombre del juego en el índice del carrusel (`self.current_index`) y construye la ruta completa al archivo ROM. Antes de iniciar un nuevo juego, verifica si ya hay un proceso de juego en ejecución (`self.current_game_process`). Si es así, muestra una notificación indicando que primero se debe cerrar el juego en ejecución. Si no hay ningún juego corriendo, inicia el emulador Mednafen en modo pantalla completa con el juego seleccionado.

```
selected_game = self.roms[self.current_index]
    rom_path = os.path.join(self.roms_directory, selected_game)
    if self.current_game_process and self.current_game_process.poll() is None:
        return
    self.current_game_process = subprocess.Popen(["mednafen", "-fs", "1", rom_path])
```

7. Configuración de la Raspberry Pi Pico

Para poder utilizar la Raspberry pi Pico como un mando de control remoto, es necesario instalar CircuitPython, lo que se detallará a continuación:

7.1. Paso 1: Descargar e instalar CircuitPython

Es necesario ingresar a la pagina oficial de CircuitPython y seleccionar nuestra tarjeta, en este caso la pico. Posteriormente, descargamos el archivo.

Una vez que descargamos nuestro archivo, procedemos a conectar la raspberry pi pico, manteniendo presionado el botón de BOOTSEL, para que sea reconocida en modo carga. Ya que conectamos la pico, se reconocer como RPI-RP2. Debemos arrastrar el archivo que descargamos anteriormente, de tal forma que automáticamente esta se reiniciará.

7.2. Paso 2: Instalación del IDE

Es necesario descargar un IDE que nos ayude a programar la pico, por lo que procedemos a instalar el programa Thonny desde su pagina oficial. Una vez que se termine de descargar procedemos a ejecutar el programa, configurando el interprete que ocuparemos. Para ello, elegimos la opción *Run* y seleccionamos *Configure interpreter*, de tal forma que se abrirá una ventana donde en el primer apartado seleccionamos *CircuitPython(generic)*, mientras que en el segundo apartado seleccionamos el puerto en el que se encuentra conectada la tarjeta. Con estas dos opciones establecidas, damos a *ok* y el programa estará configurado correctamente.

7.3. Código

El código proporcionado a continuación debe serr guardado con el nombre code.py en la raspberry pi pico. En un inicio, es necesario importar las librerías que se utilizaran para el funcionamiento, dentro de las cuales se encuentran la configuración de los pines GPIO, así como su control para configurarlos como entradas y salidas.

```
import board
import digitalio
from adafruit_hid.keyboard import Keyboard
from adafruit_hid.keycode import Keycode
import usb_hid
```

Debemos crear una instancia para configurar un teclado que se comunicará al host mediante la interfaz HID, lo que hará que la pico actúe como un teclado USB. Posteriormente, definimos cada pin del puerto GPIO a la tecla que queremos que se active al presionar el botón en ese pin.

```
keyboard = Keyboard(usb_hid.devices)

botones = [
        {"pin": board.GP6, "tecla": Keycode.TAB},
        {"pin": board.GP7, "tecla": Keycode.SPACE},
        {"pin": board.GP10, "tecla": Keycode.W},
        {"pin": board.GP12, "tecla": Keycode.C},
        {"pin": board.GP13, "tecla": Keycode.D},
        {"pin": board.GP14, "tecla": Keycode.A},
        {"pin": board.GP15, "tecla": Keycode.S},
        {"pin": board.GP27, "tecla": Keycode.BACKSPACE},
        {"pin": board.GP26, "tecla": Keycode.ENTER},
        {"pin": board.GP21, "tecla": Keycode.KEYPAD_EIGHT},
        {"pin": board.GP19, "tecla": Keycode.ESCAPE},
        {"pin": board.GP18, "tecla": Keycode.KEYPAD_SIX},
        {"pin": board.GP17, "tecla": Keycode.KEYPAD_FOUR},
        {"pin": board.GP16, "tecla": Keycode.KEYPAD_TWO},
        ]
```

Después, creamos un bucle en donde configuramos el pin de cada botón como entrada con una resistencia pull-up interna, lo que asegura que el estado del pin sea HIGH cuando el botón no está presionado.

```
for boton in botones:
    boton["objeto"] = digitalio.DigitalInOut(boton["pin"])
    boton["objeto"].switch_to_input(pull=digitalio.Pull.UP)
```

Por último, creamos un bucle que será el principal y dentro del cual en cada iteración, el programa revisa el estado de cada botón, de tal manera que si se detecta una pulsación, se llama

al método press(), lo que simula mantener presionada la tecla correspondiente. Por otro lado, no se detecta una pulsación, se llama al método release(), lo que simula soltar la tecla.

```
for boton in botones:
    boton["objeto"] = digitalio.DigitalInOut(boton["pin"])
    boton["objeto"].switch_to_input(pull=digitalio.Pull.UP)
```

8. Alambrado del circuito

El alambrado del circuito es bastante sencillo, se debe conectar un pin del botón a un puerto GPIO, mientras que el otro pin se conectará a la tierra de la pico. Es importante mencionar que los puertos GPIO a los que se conecta cada botón pueden variar a como se acomode mejor al proyecto, por lo que se debe asegurar de saber que puerto GPIO fue asignado a cada botón. El alambrado para los pines utilizados en este proyecto se puede encontrar en la Figura 1.

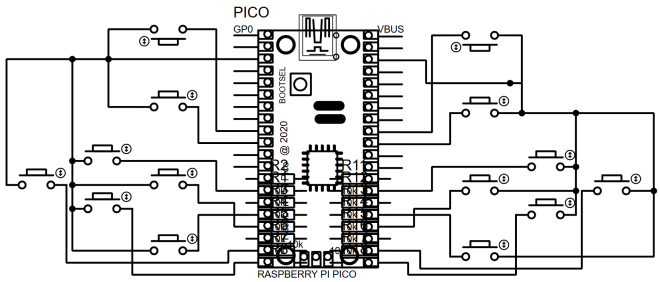

Figura 1: Alambrado del control de videojuegos mediante Raspberry pi Pico.

El video evidencia se puede consultar en el siguiente enlace.

9. Conclusiones

Los sistemas embebidos son dispositivos diseñados para llevar a cabo funciones específicas, combinando hardware y software en una plataforma compacta y eficiente. Su operación se basa en la programación de microcontroladores o microprocesadores que interactúan con sensores, actuadores y otros dispositivos, lo que permite automatizar tareas o crear interfaces funcionales. Las aplicaciones de los sistemas embebidos son variadas, abarcando desde el control de electrodomésticos y automóviles hasta la creación de dispositivos médicos y sistemas de entretenimiento.

La realización de este proyecto fue un claro ejemplo de lo que se puede lograr con un sistema embebido, ya que permitió implementar un centro de entretenimiento capaz de emular consolas retro, integrando un gamepad para controlar todas las funcionalidades sin necesidad de teclado o mouse. Este proyecto no solo demostró la versatilidad de los sistemas embebidos en la creación de experiencias de usuario personalizadas, sino que también destacó su capacidad para automatizar procesos, como la adición automática de juegos al insertar una memoria USB.

De igual forma, es importante mencionar que existieron varios retos durante la realización del código, los cuales logramos resolver a través de investigación, pruebas y ajustes. Estos desafíos incluyeron la correcta configuración del emulador para que se ejecutara automáticamente al inicio, la sincronización del gamepad con los controles del sistema, y la integración de funciones avanzadas como la animación personalizada de arranque y la detección automática de nuevos juegos.

10. Cuestionario

1. **¿Qué biblioteca permite emular un teclado USB en este código y cómo se inicializa?**

   Respuesta: La biblioteca que permite emular un teclado USB es `adafruit_hid.keyboard`. Se inicializa con la línea:

   ```
   keyboard = Keyboard(usb_hid.devices)
   ```

2. **¿Qué función tienen las resistencias pull-up configuradas para los pines GPIO en este código?**

   Respuesta: Las resistencias pull-up aseguran que el estado del pin GPIO sea **HIGH** cuando el botón no está presionado.

3. **¿Cómo define el código qué tecla del teclado corresponde a cada botón físico?**

   Respuesta: Esto se hace mediante la lista `botones`, donde cada botón está representado por un diccionario con dos claves:

   - `pin`: El pin GPIO al que está conectado el botón físico.
   - `tecla`: La tecla que se debe simular cuando el botón está presionado, representada por una constante de `Keycode`.

   Ejemplo:

   ```
   {"pin": board.GP6, "tecla": Keycode.TAB}
   ```

4. **Explica el uso de la librería pyudev en este proyecto. ¿Qué objetivo cumple en relación con los dispositivos USB?**

   Respuesta: La librería pyudev se utiliza para monitorear eventos relacionados con dispositivos USB. Específicamente, la clase GameCarouselApp la usa para detectar cuando un dispositivo USB es insertado (`action == "add"`).

5. **¿Qué función es responsable de construir la interfaz gráfica?**

   Respuesta: La función `create_widgets` es la responsable de este proceso.

6. **¿Qué hace la función `start_usb_monitoring(self)`?**

   Respuesta: Es la responsable de inicializar el monitoreo de inserciones de USB en un hilo separado.

Referencias

[1] raspberrypi, ≪*Raspberry Pi Documentation*≫ [En línea]. Disponible: https://www.raspberrypi.com/documentation/computers/getting-started.html.

[2] raspberrypi, ≪*¿Que es Raspberry Pi?*≫ [En línea]. Disponible: https://raspberrypi.cl/que-es-raspberry/.

[3] Raspberry Pi, ≪*Pico-series Microcontrollers,*≫ 2023. [En línea]. Available: https://www.raspberrypi.com/documentation/microcontrollers/pico-series.html.

[4] Raspberry Pi, ≪*Raspberry Pi pico,*≫ 2022. [En línea]. Available: https://www.raspberrypi.com/products/raspberry-pi-pico/.

[5] circuitpython, ≪*circuitpython,*≫ 2021. [En línea]. Available: https://www.raspberrypi.com/documentation/microcontrollers/pico-series.html.

[6] adafruit, ≪*¿Que es CircuitPython?,*≫ 2020. [En línea]. Available: https://learn.adafruit.com/bienvenido-a-circuitpython-2/que-es-circuitpython.