



DEPARTAMENTO DE INGENIERÍA ELÉCTRICA  
FACULTAD DE INGENIERÍA  
UNIVERSIDAD DE CONCEPCIÓN  
CONCEPCIÓN, CHILE.



# 549253 - TIC 1

## Miniproyecto 1

*Profesor: Vincenzo Caro Fuentes  
Integrantes: Matthias Aliaga Febres*

*Alan Haro Cárdenas*

*5 de octubre de 2025*

Concepción, 5 de octubre de 2025

# Índice

<b>1. Introducción</b>	<b>1</b>
<b>2. Actividad n°1: Ahorcado</b>	<b>2</b>
2.1. Ítem 1 . . . . .	2
2.2. Ítem 2 . . . . .	3
2.3. Solución . . . . .	4
2.3.1. Ítem 1 . . . . .	4
2.3.2. Ítem 2 . . . . .	8
2.3.3. Circuito realizado . . . . .	9
<b>3. Actividad n°2: La boveda de Mr.Phantom</b>	<b>10</b>
3.1. Ítem 1 . . . . .	10
3.2. Solución . . . . .	12
<b>4. Actividad n°3: Transformando Raspberry Pi en una consola de juegos</b>	<b>15</b>
4.1. Materiales a utilizar . . . . .	15
4.2. Montaje de la carcasa NESPI . . . . .	18
4.3. Instalación del software . . . . .	21
4.4. Instalación de juegos . . . . .	24
4.5. Configuración retropie en la raspberry . . . . .	25
4.6. Instalación de juegos . . . . .	26
4.6.1. Configuración WiFi . . . . .	26
4.6.2. Instalación de juegos en la consola . . . . .	28
4.7. Configuración de la Interfaz . . . . .	30
<b>5. Conclusión</b>	<b>32</b>
<b>AnexoA: Información Adicional</b>	<b>34</b>
A.5: Tipos de LEDs . . . . .	34
A.5.0: RGB LEDs . . . . .	34
A.5.0: LEDs con circuitos integrados . . . . .	34
A.5.0: RGB LEDs . . . . .	34

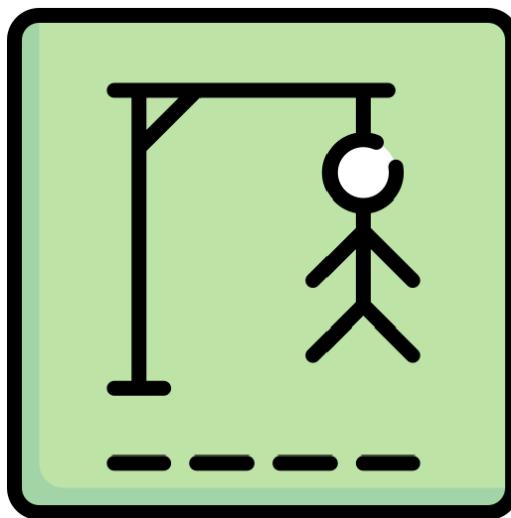
## 1. Introducción

Durante el desarrollo del curso se realizaron diversas actividades prácticas con el propósito de aprender a utilizar diferentes componentes electrónicos, tales como buzzers, botones y luces LED, a través del uso de una Raspberry Pi. Estas actividades permitieron aplicar conceptos de programación y hardware para crear sistemas interactivos. En este informe se presentan tres actividades, cada una diseñada para aplicar y reforzar los conocimientos adquiridos a lo largo del curso. La primera actividad consiste en la programación de un juego del ahorcado que integra los componentes mencionados, brindando retroalimentación visual y sonora al usuario. Las siguientes actividades continúan esta línea de trabajo, enfocándose en la resolución de problemas y en el uso correcto de los pines GPIO de la Raspberry Pi.

En conjunto, este trabajo busca demostrar la comprensión y la aplicación práctica de las tecnologías digitales mediante la programación y la integración de circuitos utilizando Python y la plataforma Raspberry Pi.

## 2. Actividad n°1: Ahorcado

El tradicional juego del ahorcado consiste en descubrir una palabra secreta adivinando sus letras una por una, dentro de un número limitado de intentos. Esta primera actividad consiste en implementar una versión digital del juego en Raspberry Pi, en la que se comenzará con un modo básico por consola, para luego evolucionar a una versión interactiva que incorpore LEDs, botones y un buzzer. La dinámica general del juego es la siguiente: al inicio, el jugador seleccionará un nivel de dificultad que determinará la extensión de las palabras a descubrir y la cantidad de intentos permitidos. Durante cada partida, el jugador ingresará o seleccionará letras para intentar completar la palabra. Cada acierto acercará al jugador a la victoria, mientras que cada error reducirá su número de intentos disponibles. El juego terminará cuando el jugador complete correctamente la palabra o bien pierda todos sus intentos.



### 2.1. Ítem 1

En este Ítem deberán preparar la base de su juego del ahorcado utilizando únicamente un teclado y la consola en Raspberry Pi. Para programar la lógica de su juego, desarrollem un programa en Python que opere de acuerdo con los siguientes pasos:

#### 1. Configuración inicial:

- El juego deberá comenzar solicitando al usuario ingresar el número de letras de las palabras a utilizar, y el número de intentos permitidos antes de perder.
- A partir de estos parámetros, el programa seleccionará aleatoriamente una palabra de una lista predefinida, ocultando sus letras con guiones bajos en pantalla.

#### 2. Desarrollo de la partida:

- El jugador ingresará letras por teclado utilizando la función `input()`.
- Si la letra pertenece a la palabra, se revelará en todas las posiciones correspondientes.
- Si la letra no pertenece, se restará un intento y se mostrará un mensaje indicando cuántos intentos quedan.
- En cada turno, la consola deberá mostrar el estado actual de la palabra, las letras descubiertas y las que faltan.

**3. Condiciones de fin de la partida:**

- El jugador ganará cuando logre descubrir toda la palabra antes de agotar los intentos.
- El jugador perderá si se queda sin intentos disponibles.
- En ambos casos, deberá mostrarse un mensaje de victoria o derrota.
- Tras terminar, el programa deberá consultar al usuario si desea iniciar una nueva partida (reiniciando la configuración) o finalizar.

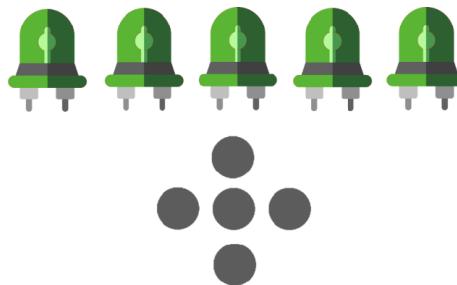
**4. Consideraciones adicionales:**

- Utilizar lo mostrado en el Anexo “Reescritura en consola” para imprimir un abecedario o la palabra oculta del juego de forma más interactiva por consola.
- Obtener la palabra a utilizar durante una partida del juego de manera dinámica desde internet.

**2.2. Ítem 2**

Utilizando como base el código del Ítem anterior, modifiquen su juego para incorporar elementos del kit de 45 sensores, utilizando al menos los siguientes módulos:

- 5 LEDs (KY-009, KY-011, KY-016 o KY-029)
- 5 botones (KY-004, KY-023, KY-040 o los entregados por el profesor).
- 1 buzzer activo (KY-012).



Con estos elementos, preparen un circuito en su protoboard que siga la estructura de la imagen anterior y desarrollen un programa en Python utilizando la biblioteca gpiozero (u otra equivalente) para controlar los LEDs, los botones y el buzzer, teniendo en cuenta las nuevas funcionalidades que se describen a continuación:

**1. Indicador visual de vidas con LEDs:**

- Los LEDs representarán el número de intentos disponibles (máximo 5).
- Se deberán definir criterios de color: por ejemplo, verde cuando se tengan todas las vidas, amarillo a la mitad, y rojo cuando quede solo 1 vida.
- A medida que se pierdan vidas, los LEDs deberán apagarse de acuerdo con el número restante (ejemplo: apagar un LED por cada intento perdido).

**2. Control de juego con botones:**

- Los 5 botones reemplazarán parcialmente la entrada por teclado, permitiendo navegar por las letras del alfabeto:

- Botón 1: avanzar una letra.
- Botón 2: retroceder una letra.
- Botón 3: avanzar 5 letras.
- Botón 4: retroceder 5 letras.
- Botón 5: seleccionar la letra actual.
- Al seleccionar, el programa validará si la letra pertenece o no a la palabra y actuará en consecuencia.

### 3. Señales sonoras con buzzer:

- Emitir un tono personalizado cada vez que se pierda una vida.
- Emitir un tono personalizado de victoria al completar la palabra.
- Emitir un tono personalizado de derrota cuando se acaben los intentos.

### 4. Consideraciones adicionales

- Implementar un contador de racha de victorias, que registre cuántas palabras se han descubierto consecutivamente sin perder. La racha deberá guardarse en un archivo y mostrarse tanto al inicio de cada juego como al finalizar una partida.

## 2.3. Solución

Primero, se creó la base del juego del ahorcado para que funcionara en la consola. Una vez hecho esto, se comenzaron a agregar los elementos propuestos en el ítem 2 y se procedió a adaptar el código para que funcionara con los nuevos elementos. En el siguiente apartado se explicarán las partes más relevantes del código final; para ello, la explicación se dividirá en dos secciones correspondientes al ítem 1 y al ítem 2, tomando en cada caso únicamente los fragmentos solicitados en cada apartado.

### 2.3.1. Ítem 1

#### 1. Funciones utilizadas:

- (a) Función para sacar las palabras de un repositorio online:

```
def get_spanish_pokemon_name(pokemon_id):
    url = f"https://pokeapi.co/api/v2/pokemon-species/{pokemon_id}/"

    with urllib.request.urlopen(url, timeout=5) as response:
        raw_data = response.read()
        data = json.loads(raw_data)

    for item in data["names"]:
        if item["language"]["name"] == "es":
            return item["name"].lower()

    return data["name"].lower()

def get_random_spanish_pokemon(max_id=1010):
    pid = randint(1, max_id)
    return get_spanish_pokemon_name(pid)
```

(a) Nombres de pokemon

```
# Github palabras
SPANISH_WORDS_URL = "https://raw.githubusercontent.com/words/an-array-of-spanish-words/master/index.json"

spanish_words = []

def load_spanish_words():
    global spanish_words
    print("Fetching Spanish word list from the internet...")
    with urllib.request.urlopen(SPANISH_WORDS_URL, timeout=5) as response:
        data = response.read()
        words = json.loads(data)
        spanish_words = [w.strip().lower() for w in words if isinstance(w, str) and w.strip()]

    print(f"Loaded {len(spanish_words)} words from online source!")

def get_random_spanish_word_by_length(length):
    global spanish_words

    if not spanish_words:
        load_spanish_words()

    filtered = [w for w in spanish_words if len(w) == length]

    if not filtered:
        raise ValueError(f"No Spanish words found with length {length}")

    return choice(filtered)
```

(b) Palabras en español

**Fig. 1:** Repositorio de palabras online

Tanto las palabras en español como los nombres de pokémon tienen asociadas dos funciones, estas funcionan de la siguiente manera:

- 1) Función `load_spanish_words()`

- Comprueba la conexión a la URL definida en *SPANISHWORDSRURL* usando *urllib.request.urlopen*.
  - Descarga el contenido del archivo JSON y lo guarda en la variable *data*.
  - Convierte ese contenido en una lista de palabras con *json.loads(data)*.
  - Procesa cada palabra para:
    - Eliminar espacios en blanco con *strip()*
    - Convertir a minúsculas con *lower()*.
    - Asegurar que el elemento es de tipo str.
    - El resultado se guarda en la lista global *spanish\_words*.
  - Finalmente, imprime cuántas palabras fueron cargadas exitosamente.
- 2) Función *get\_random\_spanish\_word\_by\_length(length)*.
- Verifica si la lista *spanish\_words* está vacía. En caso afirmativo, llama a *load\_spanish\_words()* para llenarla.
  - Filtra las palabras cuya longitud (*len(w)*) coincide exactamente con el valor del parámetro *length*. El resultado se almacena en la lista *filtered*.
  - Si no se encuentra ninguna palabra (*if not filtered:*), lanza un error con *raise ValueError*.
  - En caso contrario, selecciona una palabra aleatoria de *filtered* usando *choice()* y la devuelve.
- 3) Función *get\_spanish\_pokemon\_name(pokemon\_id)*
- Construye una URL para acceder a los datos de un Pokémon específico, usando su número de Pokédex (*pokemon\_id*).
  - Descarga la información desde la API *pokeapi.co* y la convierte en un objeto Python con *json.loads(raw\_data)*.
  - Recorre la sección *data["names"]*, que contiene los nombres del Pokémon en diferentes idiomas.
  - Si encuentra un nombre donde el campo "language"["name"] es ".es", lo devuelve en minúsculas.
  - Si no lo encuentra, devuelve el nombre por defecto (*data["name"]*).
- 4) Función *get\_random\_pokemon(max\_id = 1010)*
- Genera un número aleatorio (*pid*) entre 1 y el máximo permitido (*max\_id, por defecto 1010*).
  - Llama a *get\_spanish\_pokemon\_name(pid)* para obtener el nombre en español de ese Pokémon.
  - Devuelve dicho nombre como resultado.

## (b) Función para contar la racha de victorias:

```
#Cuerpo del programa
alfabeto = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm',
           'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', '-']

historial_letras = []

pos = 0
btn = 0
contador = 0
def abecedario(progreso, intentos):
    global pos, btn, contador, historial_letras

    with open("racha.txt", "w", encoding="utf-8") as f:
        f.write(str(contador))

    with open("racha.txt", "r", encoding="utf-8") as f:
        racha = f.read()

    print("Racha histórica de victorias: (racha)")
    print("Palabra actual: ", ".join(progreso))
    print("Intentos restantes: (intentos)")
    print("Letra actual: (alfabeto[pos].upper())")
    if len(historial_letras) > 0:
        print("Letras usadas hasta ahora: ", ".join(historial_letras).upper())
    print("Presione botón:")
    print("1: Siguiente | 2:Anterior | 3: +5 | 4: -5 | 5: Seleccionar")
    print("\n")

    btn1.when_pressed = press_button1
    btn2.when_pressed = press_button2
    btn3.when_pressed = press_button3
    btn4.when_pressed = press_button4
    btn5.when_pressed = press_button5
```

(a)

```
while True:
    if btn == "1":
        pos = (pos + 1) % 26
    elif btn == "2":
        pos = (pos - 1) % 26
    elif btn == "3":
        pos = (pos + 5) % 26
    elif btn == "4":
        pos = (pos - 5) % 26
    elif btn == "5":
        historial_letras.append(alfabeto[pos])
        btn = 0
        break
    print("Racha histórica de victorias: (racha)")
    print("Palabra actual: ", ".join(progreso))
    print("Intentos restantes: (intentos)")
    print("Letra actual: (alfabeto[pos].upper())")
    if len(historial_letras) > 0:
        print("Letras usadas hasta ahora: ", ".join(historial_letras).upper())
    print("Presione botón:")
    print("1: Siguiente | 2:Anterior | 3: +5 | 4: -5 | 5: Seleccionar")
    print("\n")
    btn = 0
```

(b)

**Fig. 2: Función .<sup>a</sup>bebedario”**

Aunque esta función contiene la lógica para navegar por el alfabeto y seleccionar letras, en el Ítem 1 solo se considera la parte que gestiona la variable racha.

- 1) Al inicio, la función crea y abre el archivo racha.txt y obtiene el valor registrado, que representa la racha histórica de victorias. Dicho valor se almacena en la variable racha.
  - 2) Cada vez que el jugador gana (según lo definido en dificultad()), la variable contador aumenta en 1 y se escribe este nuevo valor en racha.txt.
  - 3) En caso de derrota, la variable contador se reinicia en 0 y se sobrescribe el archivo racha.txt, asegurando que la racha de victorias refleje fielmente el desempeño del jugador.
- (c) Función para elegir la dificultad:

```
def dificultad(palabra_objetivo):
    intentos = validador(1,5,"Número de intentos (máximo 5): ")
    print("\n")
    progreso = ["_"]*len(palabra_objetivo)
    contador = 0
    while intentos > 0:
        if intentos == 5:
            red.off()
            green.off()
            blue.on()
        elif intentos == 3 or intentos == 4:
            red.off()
            green.on()
            blue.off()
        else:
            red.on()
            green.off()
            blue.off()
        abecedario(progreso,intentos)
        letra_candidata = alfabeto[pos]
        if letra_candidata in palabra_objetivo and " ".join(progreso) != palabra_objetivo:
            print("Fantástico! Sigue así!")
            melodía_buzzer("letra_correcta")
            for i in range(len(palabra_objetivo)):
                if palabra_objetivo[i] == letra_candidata:
                    progreso[i] = letra_candidata
```

(a)

```
abecedario(progreso,intentos)
letra_candidata = alfabeto[pos]
if letra_candidata in palabra_objetivo and " ".join(progreso) != palabra_objetivo:
    print("Fantástico! Sigue así!")
    melodía_buzzer("letra_correcta")
    for i in range(len(palabra_objetivo)):
        if palabra_objetivo[i] == letra_candidata:
            progreso[i] = letra_candidata
else:
    intentos -= 1
    print("No te rindas! Por favor, continua")
    melodía_buzzer("letra_incorrecta")

if " ".join(progreso) == palabra_objetivo:
    print("Has ganado! La palabra era: (palabra_objetivo).")
    melodía_buzzer("victoria")
    break

if intentos == 0:
    contador = 0
    with open("racha.txt", "w", encoding="utf-8") as f:
        f.write(str(contador))

    print("Has perdido! La palabra era: (palabra_objetivo). No te deprimas! Vuelve a intentarlo")
    melodía_buzzer("derrota")
```

(b)

**Fig. 3: Función ”dificultad”**

Esta función organiza el flujo principal de la partida. Sus pasos son los siguientes:

- 1) Solicita al usuario un número de intentos dentro del rango permitido (1 a 5) mediante la función validador(). El resultado se guarda en la variable intentos.
- 2) Inicializa la variable progreso, la cual contiene una lista de guiones bajos (“\_”) que representan las letras aún no adivinadas de la palabra objetivo.
- 3) Establece el ciclo principal del juego, que se mantiene activo mientras intentos ≠ 0. Dentro de este ciclo:
  - La letra seleccionada se almacena en *letra\_candidata*. (La forma de elegir la letra se explicará en el apartado 2)

- Si *letra\_candidata* se encuentra en la palabra objetivo (*palabra\_objetivo*), se actualiza la lista progreso reemplazando los guiones bajos por la letra correcta.
  - Si no se encuentra, se descuenta 1 de la variable intentos.
- 4) El ciclo se interrumpe en dos posibles casos:
- Si `"".join(progreso)` coincide con *palabra\_objetivo*, el jugador gana. En ese momento, la variable contador se incrementa en 1 y se actualiza el archivo racha.txt.
  - Si intentos llega a 0, el jugador pierde y contador se reinicia a 0, sobrescribiendo también racha.txt.

## 2. Parte principal del código:

```
#Interfaz del usuario
aux = False
while aux == False:
    modo = str(input("SELECCIONE MODO DE JUEGO (POKEMON o NORMAL): ")).lower()
    if modo == "pokemon":
        palabra_objetivo = get_random_spanish_pokemon()
        dificultad(palabra_objetivo)
        jugar_de_nuevo = str(input("Desea empezar una nueva partida: ")).lower()
        if jugar_de_nuevo == "si":
            historial_letras = []
            pos = 0
            aux = False
        else:
            aux = True
    elif modo == "normal":
        try:
            longitud_palabra = int(input("Cantidad de letras deseada en la palabra: "))
            palabra_objetivo = get_random_spanish_word_by_length(longitud_palabra)
            dificultad(palabra_objetivo)
            jugar_de_nuevo = str(input("Desea empezar una nueva partida: ")).lower()
            if jugar_de_nuevo == "si":
                historial_letras = []
                pos = 0
                aux = False
            else:
                aux = True
        except ValueError as ve:
            print(ve)
        except Exception as e:
            print("Error fetching word:", e)
    else:
        print("MODO DE JUEGO INVALIDO. POR FAVOR, SELECCIONA 'POKEMON' o 'NORMAL'")
```

**Fig. 4:** Parte principal del código

En la parte final del código se ofrecen dos opciones al usuario: POKEMON o NORMAL. Según la elección realizada, el programa seleccionará aleatoriamente una palabra de la lista general de palabras en español o bien el nombre de un Pokémon en español, para así iniciar la partida del juego.

### 2.3.2. Ítem 2

#### 1. Funciones utilizadas:

##### (a) Función para elegir las letras con los botones:

```
#Código del programa
alfabeto = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'l', 'm',
           'n', 'ñ', 'ó', 'p', 'q', 'r', 's', 't', 'ú', 'v', 'w', 'x', 'y', 'z', '-']

historial_letras = []

pos = 0
btn = 0
contador = 0

def abecedario(progreso, intentos):
    global pos, btn, contador, historial_letras

    with open("racha.txt", "w", encoding="utf-8") as f:
        f.write(str(contador))

    with open("racha.txt", "r", encoding="utf-8") as f:
        racha = f.read()

    print("Racha histórica de victorias: (racha) ")
    print("Palabra actual: ", ''.join(progreso))
    print("Intentos restantes: (intentos)")
    print("Letra actual: (alfabeto[pos].upper())")
    if len(historial_letras) > 0:
        print("Letras usadas hasta ahora: ", ''.join(historial_letras).upper())
    print("Presione botón: ")
    print("1: Siguiente | 2:Anterior | 3: +5 | 4: -5 | 5: Seleccionar")
    print("\n")

    btn1.when_pressed = press_button1
    btn2.when_pressed = press_button2
    btn3.when_pressed = press_button3
    btn4.when_pressed = press_button4
    btn5.when_pressed = press_button5
```

(a)

```
while True:
    if btn == 0:
        sleep(1)
    else:
        if btn == "1":
            pos = (pos + 1) % 26
        elif btn == "2":
            pos = (pos - 1) % 26
        elif btn == "3":
            pos = (pos + 5) % 26
        elif btn == "4":
            pos = (pos - 5) % 26
        elif btn == "5":
            historial_letras.append(alfabeto[pos])
            btn = 0
            break
    print("Racha histórica de victorias: (racha)")
    print("Palabra actual: ", ''.join(progreso))
    print("Intentos restantes: (intentos)")
    print("Letra actual: (alfabeto[pos].upper())")
    if len(historial_letras) > 0:
        print("Letras usadas hasta ahora: ", ''.join(historial_letras).upper())
    print("Presione botón: ")
    print("1: Siguiente | 2:Anterior | 3: +5 | 4: -5 | 5: Seleccionar")
    print("\n")
    btn = 0
```

(b)

**Fig. 5: Función „abecedario”**

Función encargada de controlar la selección de letras mediante los botones, su proceso es el siguiente:

- 1) Se crea un alfabeto con todo el abecedario en español y el simbolo ”, este es usado para algunos pokemones.
- 2) Se asignan funciones a los botones físicos (btn1 a btn5) para que cada uno modifique la posición pos dentro del alfabeto o seleccione la letra actual.
- 3) Dentro de un bucle while True:
  - Si no se ha presionado ningún botón (btn == 0), espera 1 segundo.
  - Según el botón presionado:
    - Botón 1: avanza una letra (pos + 1 % 28).
    - Botón 2: retrocede una letra (pos - 1 % 28).
    - Botón 3: avanza 5 letras (pos + 5 % 28).
    - Botón 4: retrocede 5 letras (pos - 5 % 28).
    - Botón 5: selecciona la letra actual y la agrega a *historial\_letras*.
  - Después de cada acción, se actualiza la pantalla con el estado actual y se reinicia btn = 0.

(b) Función para cambiar el color del LED y encender el buzzer:

```
def dificultad(palabra_objetivo):
    intentos = validador(1,5,"Número de intentos (máximo 5): ")
    print("\n")
    progreso = ["_"]*len(palabra_objetivo)
    contador = 0
    while intentos > 0:
        if intentos == 5:
            red.off()
            green.off()
            blue.on()
        elif intentos == 3 or intentos == 4:
            red.off()
            green.on()
            blue.off()
        else:
            red.on()
            green.off()
            blue.off()
        abecedario(progreso,intentos)
        letra_candidata = alfabeto[pos]
        if letra_candidata in palabra_objetivo and "".join(progreso) != palabra_objetivo:
            print("Fantástico! Sigue así!")
            melodía_buzzer("letra_correcta")
            for i in range(len(palabra_objetivo)):
                if palabra_objetivo[i] == letra_candidata:
                    progreso[i] = letra_candidata
```

(a)

```
abecedario(progreso,intentos)
letra_candidata = alfabeto[pos]
if letra_candidata in palabra_objetivo and "".join(progreso) != palabra_objetivo:
    print("Fantástico! Sigue así!")
    melodía_buzzer("letra_correcta")
    for i in range(len(palabra_objetivo)):
        if palabra_objetivo[i] == letra_candidata:
            progreso[i] = letra_candidata
else:
    intentos -= 1
    print("No te rindas! Por favor, continua")
    melodía_buzzer("letra_incorrecta")

if "".join(progreso) == palabra_objetivo:
    contador += 1
    with open("racha.txt", "a", encoding="utf-8") as f:
        f.write(str(contador))
    print("¡Has ganado la palabra era: (palabra_objetivo)!")
    melodía_buzzer("victoria")
    break

if intentos == 0:
    contador = 0
    with open("racha.txt", "w", encoding="utf-8") as f:
        f.write(str(contador))

    print("¡Has perdido! La palabra era: (palabra_objetivo). No te deprimes! Vuelve a intentarlo")
    print("\n")
    melodía_buzzer("derrota")
```

(b)

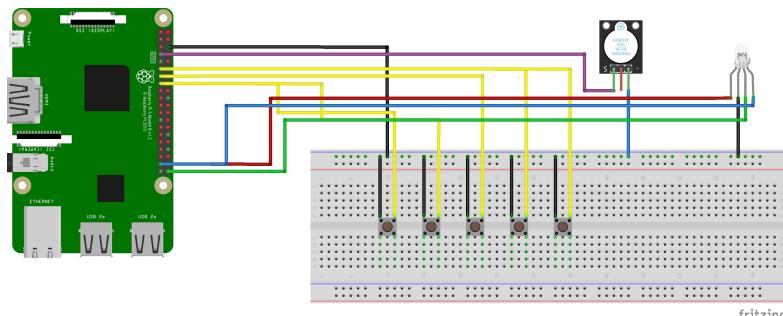
Fig. 6: Función "dificultad"

Esta función se encarga de variar los colores del LED y el sonido del Buzzer dependiendo de las acciones del jugador de la siguiente manera:

- 1) Control de LEDs según la cantidad de intentos:
  - 5 intentos: LED azul encendido, verde y rojo apagados.
  - 3-4 intentos: LED verde encendido, azul y rojo apagados.
  - 1-2 intentos: LED rojo encendido, azul y verde apagados.
- 2) Llama a la función abecedario para permitir que el jugador seleccione letras usando los botones.
- 3) Comprobación de la letra seleccionada (*letra\_candidata*):
  - Si la letra está en la palabra, se actualiza progreso y se reproduce un tono de letra correcta con el buzzer (*melodía\_buzzer("letra\_correcta")*).
  - Si la letra no está, se resta un intento y se reproduce un tono de letra incorrecta (*melodía\_buzzer("letra\_incorrecta")*).
- 4) Condiciones de fin de partida:
  - Si el jugador completa la palabra, incrementa contador, actualiza la racha en racha.txt y reproduce tono de victoria (*melodía\_buzzer("victoria")*).
  - Si se acaban los intentos, reinicia contador, actualiza racha.txt y reproduce tono de derrota (*melodía\_buzzer("derrota")*).

### 2.3.3. Circuito realizado

Finalmente, la siguiente imagen muestra un esquema del circuito realizado para resolver esta actividad.



### 3. Actividad n°2: La bóveda de Mr.Phantom

En esta segunda actividad se enfrentarán a Mr. Phantom, un ladrón experto que ha colocado un sistema de seguridad en su propia bóveda, protegiéndola con un láser y una cerradura lógica. El juego comienza al interrumpir el haz de luz del láser, donde un contador comienza a correr y el jugador debe desbloquear la cerradura lógica introduciendo una secuencia correcta de 3 botones bajo la presión de tiempo.



#### 3.1. Ítem 1

En este ítem deberán implementar la cerradura con lógica de secuencia de la bóveda de Mr. Phantom, considerando los siguientes módulos de su kit de 45 sensores:

- 3 botones (KY-004, KY-023, KY-040 o equivalentes).
- 3 LEDs (KY-009, KY-011, KY-016 o KY-029) — uno por botón.
- 1 buzzer pasivo (KY-006).
- 1 módulo láser (KY-008)
- 1 fotorresistencia (KY-018).

Con estos elementos, preparen un circuito en su protoboard que siga la estructura de la imagen anterior, donde cada LED estará asociado con un único botón que permitirá al jugador interactuar manualmente con los elementos de la cerradura lógica. Además, el módulo láser deberá apuntar en todo momento a la fotorresistencia.

Nota: Las fotorresistencias son muy sensibles a la luz ambiental, por lo que se recomienda ubicarla en un lugar oscuro, como una caja.

Luego, para programar la lógica de su juego en Raspberry Pi, desarrolle un programa en Python que se guíe por los siguientes pasos:

### 1. Configuración inicial

- El juego deberá comenzar solicitando al usuario ingresar el tiempo total del juego y si se permite o no la repetición de un botón dentro de la secuencia.
- A partir de estos parámetros, el programa debe generar aleatoriamente una secuencia de 3 pasos (p. ej., Botón 1 → Botón 3 → Botón 2), y se esperará a la acción de inicio del juego.

### 2. Inicialización por láser

- El juego inicia automáticamente al detectar que se interrumpe el haz del láser (en el momento cuando la fotorresistencia cambia de estado).
- Se activa un temporizador que define el tiempo total del juego.

### 3. Desarrollo de la partida

- Cada botón está asociado a un LED, donde el LED se debe mantener encendido mientras el botón está presionado.
- Si se presiona el botón correcto según el orden actual, avanza al siguiente paso de la secuencia, y el buzzer reproduce un tono corto de acierto.
- Si se presiona un botón que está fuera del orden de la secuencia, se reinicia el progreso de la secuencia a su primer elemento, se descuenta 1 segundo del tiempo restante y el buzzer reproduce un tono corto de fallo.
- El tiempo restante debe mostrarse y actualizarse en consola (p. ej., “Tiempo restante: 8.3 s”).

### 4. Cierre de la partida

- Si se completa la secuencia dentro del tiempo establecido, se debe reproducir una melodía de victoria por el buzzer pasivo.
- Si el tiempo se agota, se debe reproducir una melodía de game over por el buzzer pasivo.
- Tras terminar, el programa deberá consultar al usuario si desea iniciar una nueva partida (reiniciando la configuración desde el paso 1) o finalizar.

### 5. Consideraciones adicionales

- La consola debe informar de cada paso o movimiento que transcurre durante el juego.
- Utilizar lo mostrado en el Anexo “Reescritura en consola” para imprimir un tiempo interactivo por consola.
- Agregarle dificultad adicional al juego al permitir que las secuencias cambien durante el tiempo. Por ejemplo:
  - 10 s con 0 cambios (1 sola secuencia)
  - 20 s con 1 cambio (2 secuencias totales).
  - 30 s con 2 cambios (3 secuencias totales).

De esta forma, una vez transcurrido el intervalo de cambio, se debe generar una nueva secuencia y volver el progreso del juego al paso 3.

### 3.2. Solución

Para resolver esta actividad, se optó por no utilizar un láser, dado que no se encontró una manera viable de implementarlo. En su lugar, el juego se iniciará al tocar la fotorresistencia. El código funciona de la siguiente manera:

#### 1. Asignación de pines GPIO a sensores y actuadores

- Se definen los LEDs, botones, buzzer y fotorresistencia con sus pines GPIO correspondientes:
  - *led\_rojo, led\_verde, led\_azul* → LEDs indicadores de botones.
  - *btn\_rojo, btn\_verde, btn\_azul* → Botones que el jugador debe presionar para introducir la secuencia.
  - *foto* → Fotorresistencia usada para iniciar el juego.
  - *buzzer* → TonalBuzzer que reproduce sonidos según el estado del juego.

#### 2. Generación de la clave del juego (*modo\_juego*)

- Se crean dos listas de posibles secuencias de colores:
  - *claves\_con\_repeticion* → Secuencias que permiten repetir colores.
  - *clave\_sin\_repeticion* → Secuencias que no permiten repetir colores.
- Dependiendo de la respuesta del usuario (repeticion), se elige aleatoriamente una secuencia de una de estas listas usando choice.

#### 3. Reproducción de melodías (*toca\_melodia*)

- Se definen melodías distintas según la situación:
  - ”*boton\_correcto*” → Se toca un acorde positivo al presionar el botón correcto.
  - ”*boton\_incorrecto*” → Se toca un acorde distinto si el botón es incorrecto.
  - ”*derrota*” → Melodía de derrota al fallar la secuencia.
  - ”*victoria*” → Melodía de victoria al acertar toda la secuencia.
- Cada melodía se reproduce usando *buzzer.play(nota)* y *sleep(tempo[i])* para controlar la duración de cada nota.

#### 4. Comparación de botones presionados con la clave (comparador)

- Esta función compara los botones presionados por el jugador (*lista\_btn*) con la secuencia de la clave (clave).
- Se usan etapas (contador) para controlar el progreso:
  - (a) Primer botón ( → ”*paso1\_completado*”)
  - (b) Segundo botón (”*paso1\_completado*” → ”*paso2\_completado*”)
  - (c) Tercer botón (”*paso2\_completado*” → *ganaste = Truesiescorrecto*)
- Si un botón es incorrecto:
  - Se reproduce la melodía de error (*toca\_melodia(”boton\_incorrecto”)*).
  - Se limpia la lista de botones (*lista\_btn.clear()*).
  - Se decrementa el tiempo restante (*tiempo\_juego - = 1*).
  - Se apagan los LEDs (*apaga\_leds()*).

## 5. Funciones de activación de LEDs (*activa\_led\_rojo/verde/azul*)

- Cada función:
  - Enciende el LED correspondiente.
  - Añade el color presionado a la lista de botones (*lista\_btn.append("color")*).
  - Llama a *comparador()* para evaluar la secuencia.

## 6. Función para apagar LEDs (*apaga\_leds*)

- Apaga los tres LEDs (*led\_rojo.off()*, *led\_verde.off()*, *led\_azul.off()*).
- Se utiliza cuando se reinicia el juego o se presiona un botón incorrecto.

## 7. Desarrollo principal del juego

- a) Se solicita al usuario el tiempo total de juego (*tiempo\_juego*) y si permite repetición de colores (*repeticion*).
- b) El juego inicia al presionar la fotorresistencia (*foto.wait\_for\_release()*).
- c) Se selecciona la clave de la secuencia con *modo\_juego(repeticion)*.
- d) Se inicializan variables:
  - *ganaste* = False → Controla si el jugador ganó.
  - *lista\_btn* = [] → Lista de botones presionados por el jugador.
  - *contador* = → Controla el progreso de la secuencia.
- e) Se asignan las funciones de activación de LEDs a los botones con *btn.when\_pressed*.
- f) Mientras no se acabe el tiempo y no se gane

*(while time() - inicio < tiempo\_juego and ganaste == False)*

se actualiza el tiempo restante en consola y se espera la interacción del jugador.

- g) Al finalizar:
  - Si *ganaste* = True, se reproduce la melodía de victoria y se pregunta si quiere volver a jugar.
  - Si no, se reproduce la melodía de derrota y se ofrece la opción de reiniciar el juego.
  - En ambos casos, se apagan los LEDs con *apaga\_leds()*.

Finalmente, la siguiente imagen muestra un esquema del circuito realizado para resolver esta actividad.

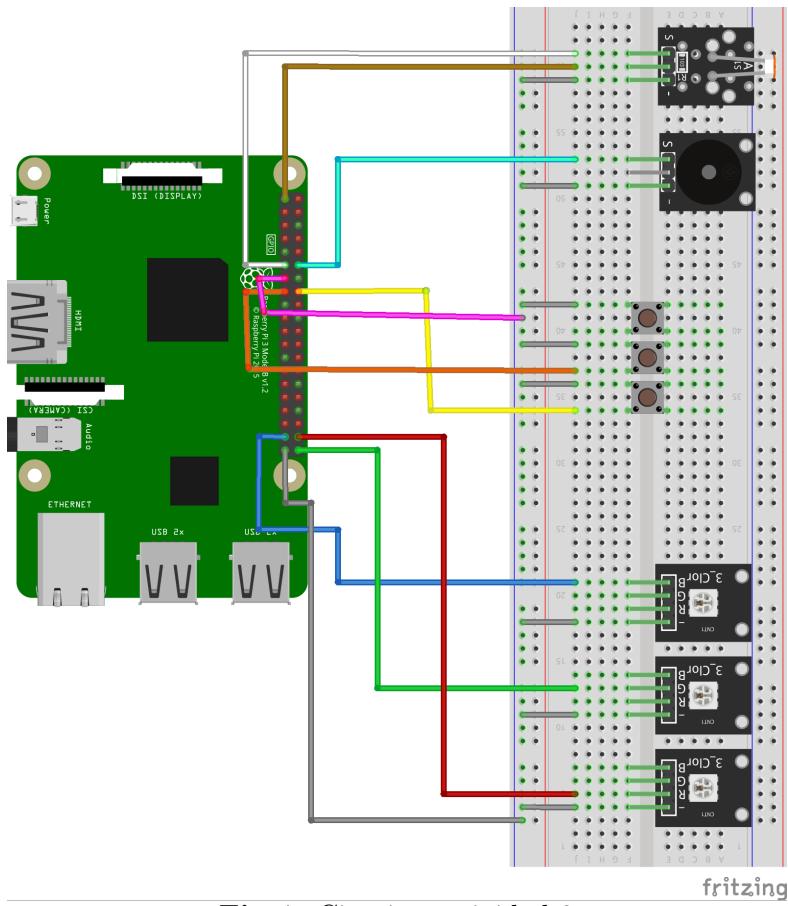
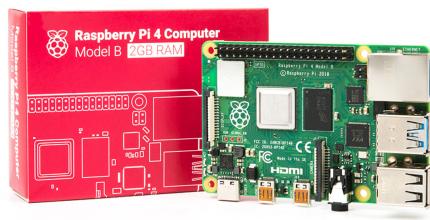


Fig. 7: Circuito actividad 2

## 4. Actividad n°3: Transformando Raspberry Pi en una consola de juegos

### 4.1. Materiales a utilizar

1. RaspBerry PI 4B 2G



**Fig. 8:** RaspBerry PI 4B 2G. Fuente: [1]

2. Carcasa NESPI



**Fig. 9:** Carcasa NESPI. Fuente: [2]

3. Fuente Alimentación con cortador de corriente



**Fig. 10:** Fuente de alimentación. Fuente: [3]

4. Mandos a elección (nes,nintendo,etc)



**Fig. 11:** Mandos snes. Fuente: [3]

5. SD de 64GB



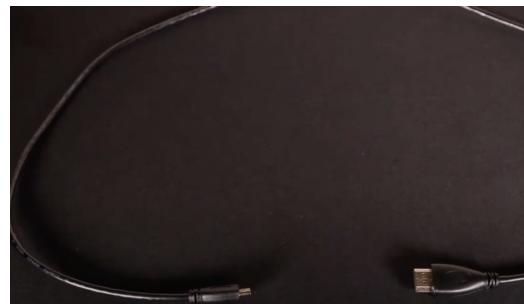
**Fig. 12:** Tarejeta micro SD. Fuente: [3]

6. Lector de tarjetas SD



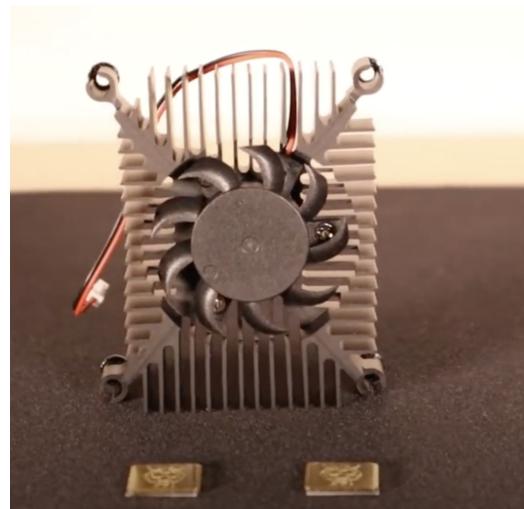
**Fig. 13:** Lector SD. Fuente: [3]

7. Cable HDMI/Micro HDMI



**Fig. 14:** Cable HDMI. Fuente: [3]

8. Ventilador + disipador



**Fig. 15:** Disipador y pasta térmica. Fuente: [3]

Para esta guía se utilizará una carcasa NESPI como la que se muestra en 9, la cual no es indispensable. Sin embargo, se recomienda ya que mejora la circulación del aire y proteje la raspberry utilizada.

#### 4.2. Montaje de la carcasa NESPI

Este procedimiento se realizará teniendo la carcasa NESPI 4 CASE como referencia. Los pasos a seguir son los siguientes:

1. Al abrir la carcasa NESPI, se aprecia un espacio destinado a la Raspberry Pi.



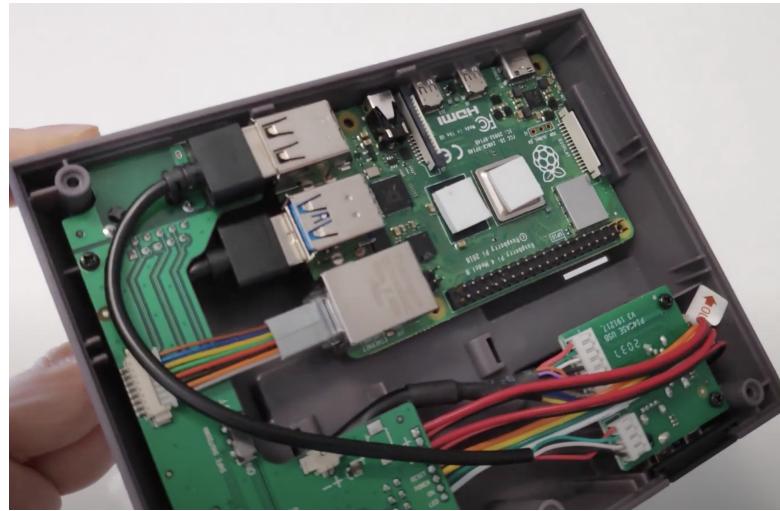
Fig. 16: Interior de la carcasa NESPI. Fuente: [4]

2. Se deben colocar las dos pastas térmicas incluidas con el disipador sobre el procesador de la Raspberry Pi.



Fig. 17: Interior de la carcasa NESPI. Fuente: [4]

3. La Raspberry Pi debe introducirse en el espacio de la carcasa NESPI mencionado anteriormente, conectando posteriormente los dos cables USB y el cable Ethernet a la placa.

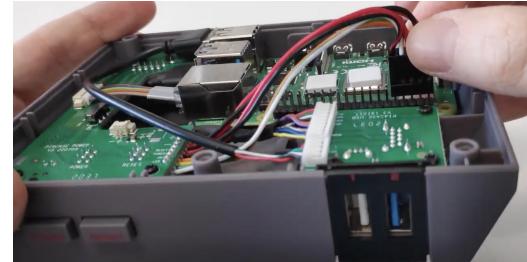


**Fig. 18:** Ensamblaje de la raspberry. Fuente: [4]

4. En el interior de la carcasa se encuentra un cable rojo, el cual debe conectarse a la Raspberry Pi según lo indicado en la imagen 19.



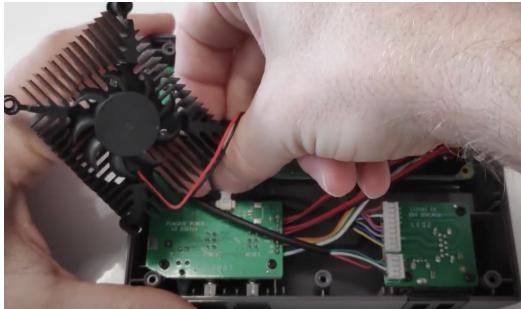
(a) Cable



(b) Conexión a la raspberry

**Fig. 19:** Conexión del cable. Fuente: [4]

5. Se debe conectar el cable del disipador a la carcasa, tal como se indica en la imagen 20, y posteriormente fijar el disipador a la Raspberry Pi usando los tornillos correspondientes. Es importante retirar el adhesivo de la pasta térmica utilizada anteriormente, en caso de que este estuviera presente.



(a) Conexión del disipador



(b) Ensamblaje del disipador

**Fig. 20:** Conexión del disipador. Fuente: [4]

6. Finalmente, se debe conectar el cable USB ubicado en la parte superior de la carcasa a la Raspberry Pi. A continuación, se procede a ensamblar completamente la carcasa NESPI.



(a) Conexión del cable USB



(b) Ensamblaje de la carcasa NESPI

**Fig. 21:** Ensamblaje de la carcasa NESPI. Fuente: [4]

#### 4.3. Instalación del software

Primero, se debe ingresar a la página oficial de Raspberry Pi a través del siguiente enlace: [Página oficial de descargas de Raspberry Pi](#). Una vez dentro, se deberán seguir los pasos que se indican a continuación:

1. Descargar Raspberry PI Imager.

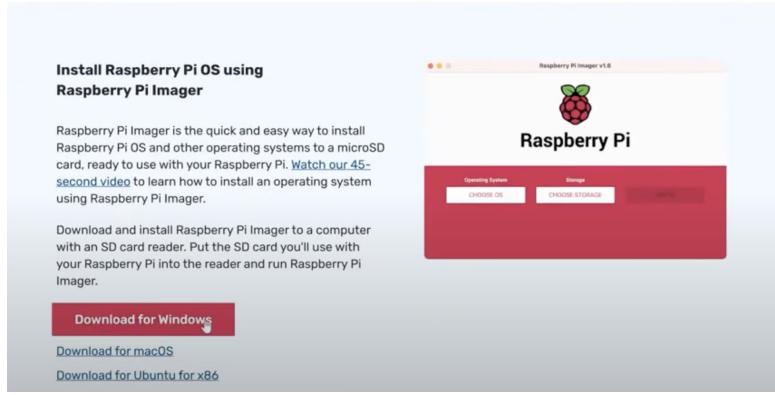


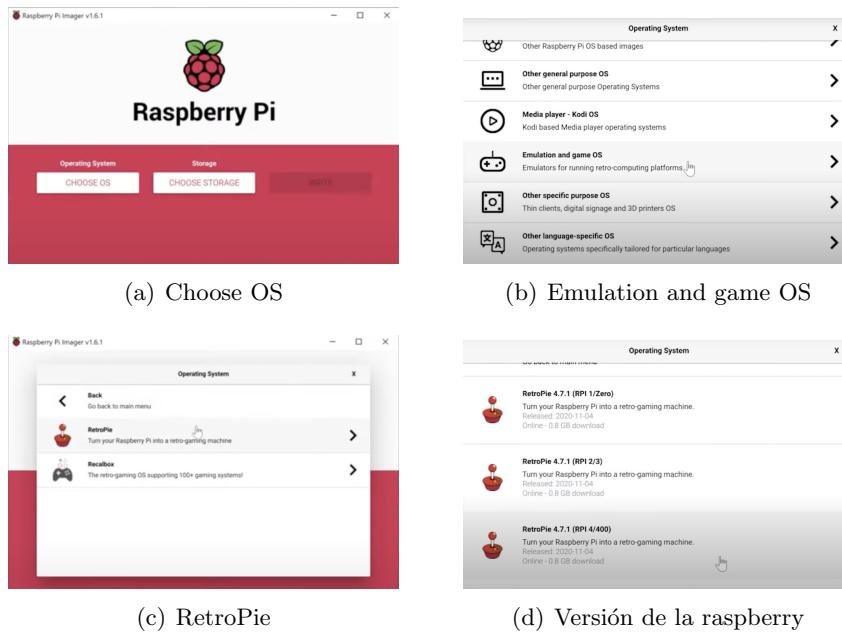
Fig. 22: RaspBerry PI Imager. Fuente: [3]

2. Ejecutar el archivo instalado e instalar el software



Fig. 23: Instalación del software. Fuente: [3]

### 3. Elegir el sistema operativo llamado RetroPie



**Fig. 24:** Elección del sistema operativo. Fuente: [3]

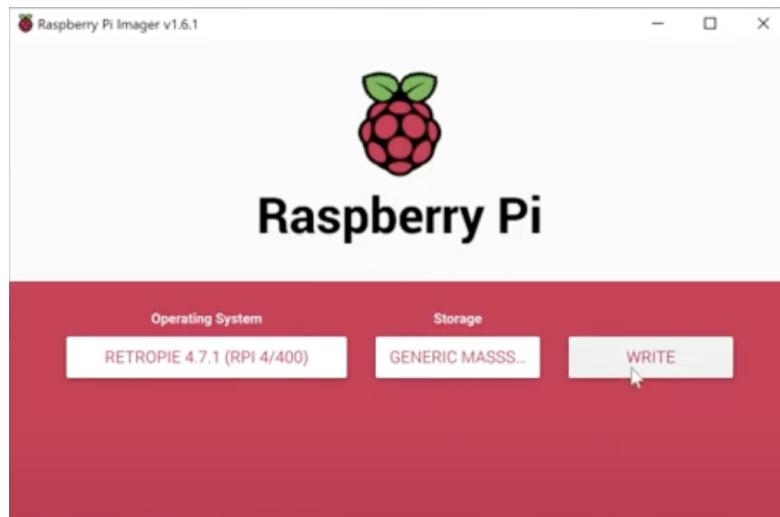
en la imagen 24(d) se debe elegir la versión dependiendo de cual Raspberry se este usando (en este caso se usa RPI4).

### 4. Se debe seleccionar el dispositivo de almacenamiento: para ello, la tarjeta SD debe insertarse en el lector SD y conectarse al computador. A continuación, se selecciona la opción “Choose Storage” y se elige la tarjeta SD previamente conectada.



**Fig. 25:** Elección del almacenamiento. Fuente: [3]

5. Finalmente, se selecciona la opción "Write".



**Fig.** 26: Instalación del software. Fuente: [3]

Una vez finalizado este proceso, se debe retirar la tarjeta micro SD del lector y colocarla en la carcasa NESPI o directamente en la Raspberry Pi.

#### 4.4. Instalación de juegos

Existen diversas páginas web para descargar ROMs; en este caso, la instalación de los juegos se realizará utilizando el sitio [wowroms.com](http://wowroms.com), al cual se puede acceder mediante el siguiente enlace: [Sitio web para descargar ROMs](#).

El procedimiento a seguir es el siguiente:

1. Buscar el nombre del juego en la barra de búsqueda.

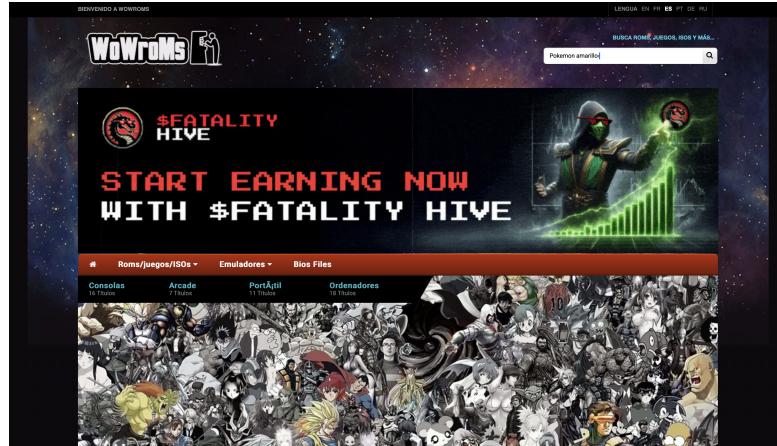


Fig. 27: Barra de búsqueda. Fuente: [5]

2. Elegir y descargar la ROM.

(a)

(b)

Fig. 28: Instalación ROM. Fuente: [5]

#### 4.5. Configuración retropie en la raspberry

Para llevar a cabo esta etapa, es necesario conectar un control a la carcasa NESPI o a la Raspberry Pi. Los pasos a seguir son los siguientes:

1. Conectar la carcasa NESPI a una pantalla usando el cable HDMI.
2. Con la tarjeta SD insertada en la carcasa NESPI, se debe conectar la carcasa a la fuente de alimentación.
3. Al encender la consola, la raspberry instalará y creará automáticamente el sistema y particiones necesarias para el correcto funcionamiento de retropie. Cuando finalice este proceso, se mostrará una pantalla para configurar el control.



**Fig. 29:** Configuración del control. Fuente: [3]

4. Dejar pulsado cualquier botón del control y empezar a configurarlo



**Fig. 30:** Configuración del control. Fuente: [3]

Es importante asignar un botón a la opción "HOTKEY ENABLE" como se muestra en la imagen 30(b), ya que esta es necesaria para acceder a funciones importantes de la consola (por ejemplo guardar, cargar y salir de partidas).

## 4.6. Instalación de juegos

Existen diversas formas de instalar juegos en la Raspberry Pi, en este ítem se explicará el procedimiento para realizar la instalación a través de una conexión WiFi. Los pasos a seguir son los siguientes:

### 4.6.1. Configuración WIFI

1. Al encender la consola, se visualizará la opción para acceder a la configuración de RetroPie. Para ello, se debe presionar el botón asignado a la función Select.

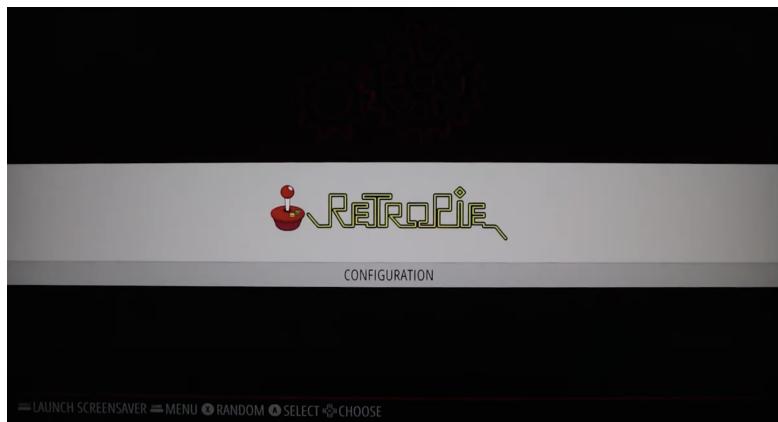


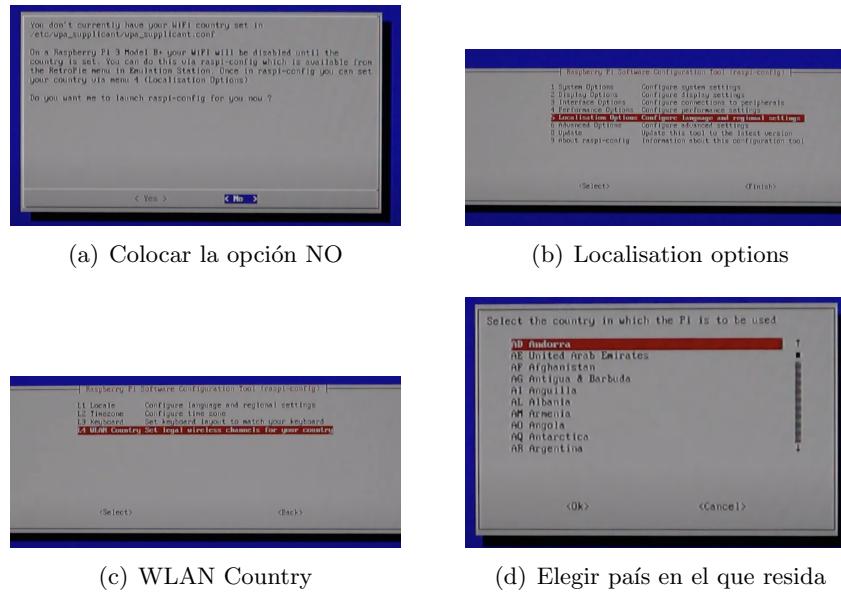
Fig. 31: Configuración RetroPie. Fuente: [3]

2. Seleccionar la opción WiFi al final del menú.



Fig. 32: Opción WiFi. Fuente: [3]

3. Seguir la siguiente secuencia:



**Fig. 33:** Selección del País. Fuente: [3]

Despues de haber hecho esto, dar a la opción Finish y reiniciar la consola.



**Fig. 34:** Configuración del control. Fuente: [3]

4. Finalmente, en la esquina superior izquierda de la pantalla se mostrará la dirección IP identificada como Wireless IP. Esta dirección deberá ser registrada para su posterior uso.

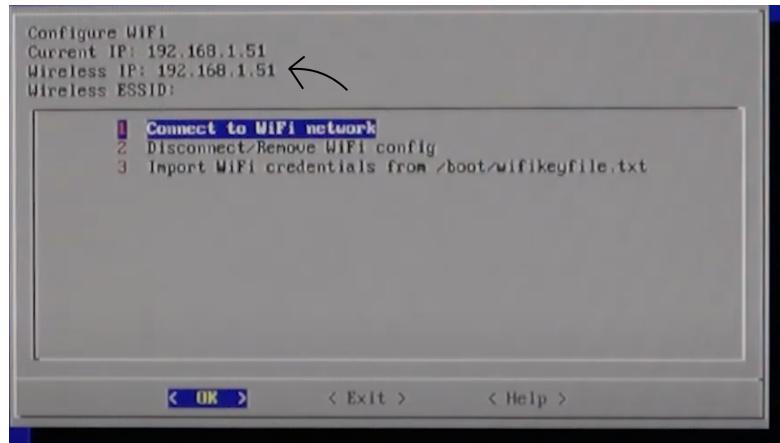


Fig. 35: Dirección IP. Fuente: [3]

#### 4.6.2. Instalación de juegos en la consola

Para llevar a cabo este proceso será necesario disponer de un computador con sistema operativo Windows. A continuación, se detallan los pasos a seguir:

1. Encender la consola. Luego, desde el escritorio de Windows, se debe presionar la combinación de teclas Ctrl + R e introducir la dirección IP anotada previamente en el siguiente formato: \\Dirección IP. Finalmente, se debe seleccionar la opción Aceptar.

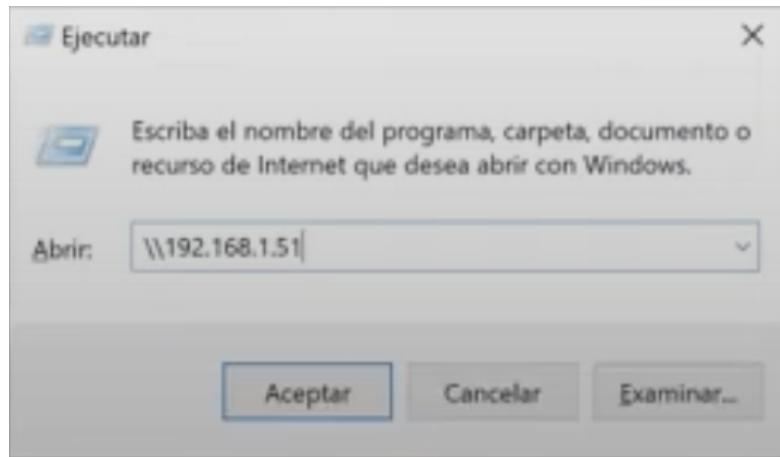
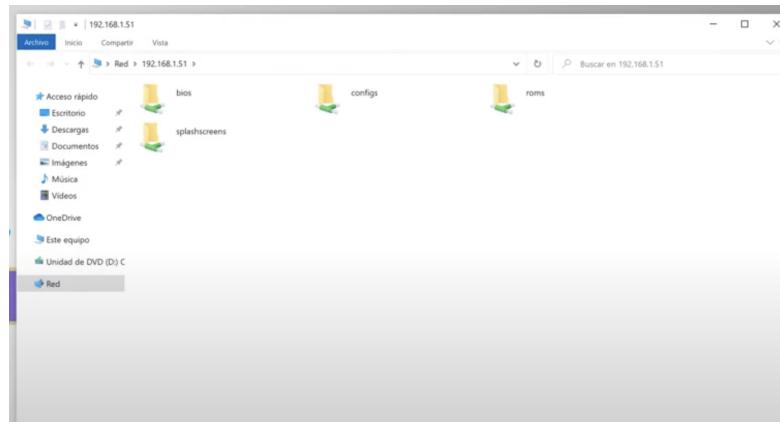


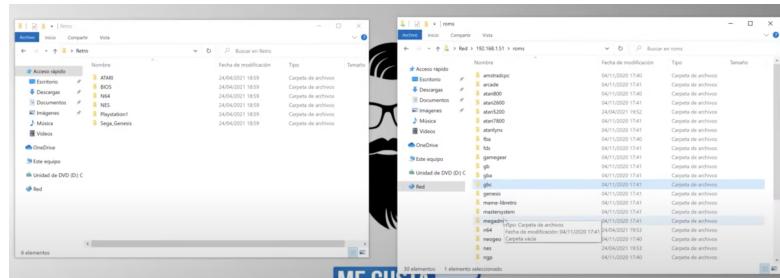
Fig. 36: Ejecutar la IP. Fuente: [6]

2. Seleccionar la carpeta roms.



**Fig. 37:** Carpeta roms. Fuente: [6]

3. Se debe seleccionar la carpeta correspondiente al emulador de la consola que se desea utilizar y copiar en ella los juegos. Es importante verificar que los juegos sean compatibles con el emulador correspondiente. Una vez hecho esto, reiniciar la consola.



**Fig. 38:** Carpeta roms. Fuente: [6]

4. Al encender la consola, se podrán visualizar los diferentes emuladores en los que se han colocado juegos. Finalmente, se debe seleccionar el emulador y el juego deseado.



(a) Selección del emulador



(b) Selección del juego

**Fig. 39:** Interfaz de los juegos. Fuente: [6]

#### 4.7. Configuración de la Interfaz

Es posible modificar la interfaz de la consola para hacerla más atractiva. A continuación, se detallan los pasos a seguir:

1. En el menú principal (este se puede apreciar en la imagen 39(a)) pulsar el botón asignado a la función Start. Esto abrirá el menú de RetroPie. Aquí, se deberá colocar la opción SCRAPER.

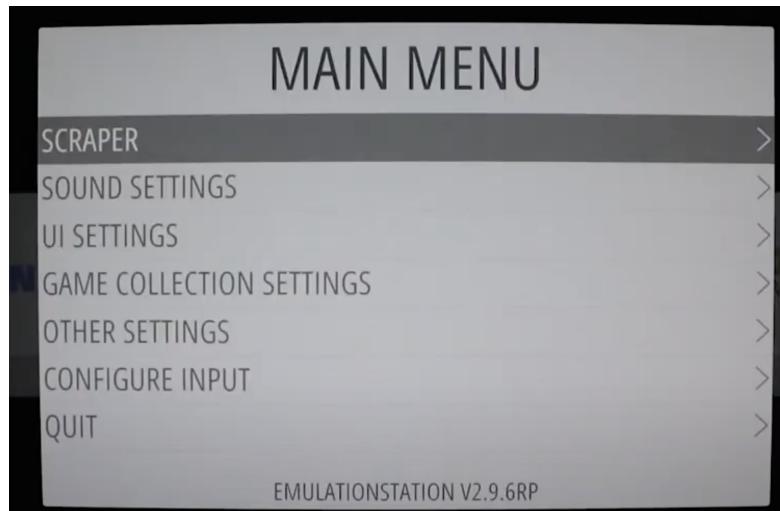


Fig. 40: Menú de RetroPie. Fuente: [6]

2. Seguir la siguiente secuencia:

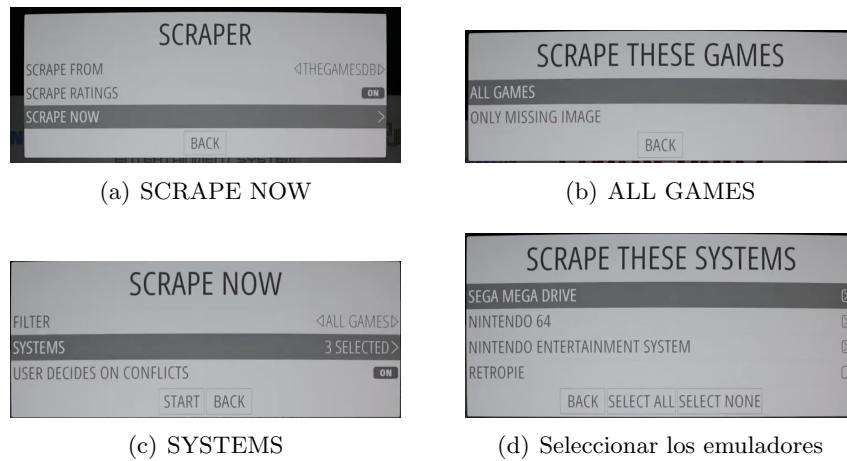


Fig. 41: Configuración de la interfaz. Fuente: [6]

Despues de haber hecho esto, se deberá colocar la opción USED DECIDES ON CONFLICTS en off (ver imagen 41(c)).

3. Finalmente, seleccionar la opción start. Una vez hecho todo este proceso, los juegos tendrán una imagen y descripción asociadas, como se puede observar en la figura 42.

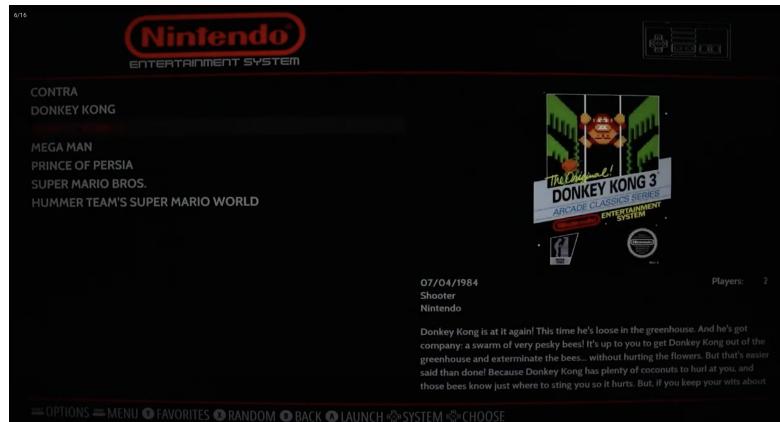


Fig. 42: Imagen de referencia. Fuente: [6]

## 5. Conclusión

A lo largo de las tres actividades realizadas, se logró aplicar de manera práctica los conocimientos adquiridos sobre el uso de la Raspberry Pi y sus periféricos. En la primera actividad, se desarrolló un juego del ahorcado incorporando elementos como LEDs, botones y un buzzer, consolidando la comprensión de la interacción entre hardware y software. La segunda actividad permitió profundizar en la implementación de sistemas de seguridad mediante la programación de secuencias lógicas con sensores, botones y LEDs, fortaleciendo habilidades de lógica digital y control de dispositivos. Finalmente, en la tercera actividad se exploró la creación de una consola de juegos retro con RetroPie, lo que facilitó la integración de software y hardware para emular consolas clásicas y comprender la configuración de sistemas complejos.

En conjunto, estas actividades permitieron reforzar la capacidad de planificar, implementar y probar proyectos interactivos con la Raspberry Pi, promoviendo tanto el pensamiento lógico como la creatividad en el uso de sensores y actuadores.

## Referencias

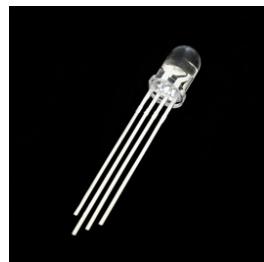
- [1] P. Electric, “Imagen raspberry pi 4b,” <https://cdn11.bigcommerce.com/.../RaspberryPi4B.jpg>, 2025, accedido el 22 de septiembre de 2025.
- [2] Starware, “Imagen nespi,” <https://tienda.starware.com.ar/wp-content/uploads/2021/09/gabinete-carcasa-retroflag-nespi-4-case-raspberry-pi-4-2375-4579.jpg>, 2025, accedido el 22 de septiembre de 2025.
- [3] SysBeards, “Como instalar retropie en raspberry pi 4,” <https://www.youtube.com/watch?v=X-UD7Xmrl5w&t=300s>, 2025, accedido el 22 de septiembre de 2025.
- [4] G. Gamer, “Como instalar retropie en raspberry pi 4,” <https://www.youtube.com/watch?v=S0-oJ6S8sSc>, 2025, accedido el 22 de septiembre de 2025.
- [5] WoWroms, “Wowroms,” <https://wowroms.com/es/>, 2025, accedido el 22 de septiembre de 2025.
- [6] SysBeards, “Como pasar juegos a retropie — como configurar retropie,” [https://www.youtube.com/watch?v=qY7GvgmeYI0&list=PL18-KiZG\\_nIMiqJ2Ex8ijsEUNBM2A0WBI&index=6](https://www.youtube.com/watch?v=qY7GvgmeYI0&list=PL18-KiZG_nIMiqJ2Ex8ijsEUNBM2A0WBI&index=6), 2025, accedido el 22 de septiembre de 2025.

## Anexo A: Información Adicional

### A.5: Tipos de LEDs

#### A.5.0: RGB LEDs

¡Los LED RGB (Rojo-Verde-Azul) son en realidad tres LED en uno! Pero eso no significa que solo pueda hacer tres colores. Debido a que el rojo, el verde y el azul son los colores primarios aditivos, puede controlar la intensidad de cada uno para crear todos los colores del arco iris. La mayoría de los LED RGB tienen cuatro pines: uno para cada color y un pin común. En algunos, el pin común es el ánodo, y en otros, es el cátodo.



**Fig. 43:** LED de cátodo transparente común RGB. Fuente:

#### A.5.0: LEDs con circuitos integrados

Algunos LED son más inteligentes que otros. Tomemos el LED de ciclismo, por ejemplo. Dentro de estos LED, en realidad hay un circuito integrado que permite que el LED parpadee sin ningún controlador externo. Aquí hay un primer plano del IC (el gran chip cuadrado negro en la punta del yunque) que controla los colores.

¡Simplemente enciéndalo y míralo ir! Estos son excelentes para proyectos en los que desea un poco más de acción, pero no tiene espacio para circuitos de control. ¡Incluso hay LED parpadeantes RGB que recorren miles de colores!

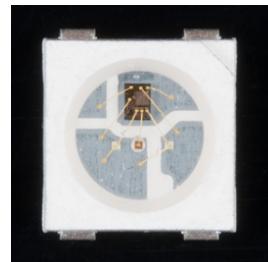


**Fig. 44:** Primer plano del LED de ciclo lento de 5 mm. Fuente:

#### A.5.0: RGB LEDs

Los LED SMD no son tanto un tipo específico de LED sino un tipo de paquete. A medida que la electrónica se hace cada vez más pequeña, los fabricantes han descubierto cómo meter más componentes en un espacio más pequeño. Las piezas SMD (dispositivo de montaje en superficie) son versiones pequeñas de sus contrapartes estándar. Aquí hay un primer plano de un LED direccionable WS2812B empaquetado en un pequeño paquete 5050.

Los LED SMD vienen en varios tamaños, ¡desde bastante grandes hasta más pequeños que un grano de arroz! Debido a que son tan pequeños y tienen almohadillas en lugar de piernas, no son tan fáciles de trabajar, pero si tiene poco espacio, podrían ser justo lo que recetó el médico.



**Fig. 45:** Primer plano direccional WS2812B. Fuente: