



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



549253 - TIC 1

Miniproyecto 2

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

Alan Haro Cárdenas

22 de noviembre de 2025

Concepción, 22 de noviembre de 2025

Índice

1. Introducción	1
2. Actividad n°1: TIC is Among Us	2
2.1. Ítem 1	3
2.2. Ítem 2	3
2.3. Ítem 3	4
2.4. Ítem 4	5
2.5. Solución	6
2.5.1. Ítem 1	6
2.5.2. Ítem 2	12
2.5.3. Item 3	15
3. Actividad n°2: Programando una Pokédex	16
3.1. Ítem 1	16
3.2. Solución	17
3.2.1. Diseño	17
3.3. Funcionamiento	18
3.4. Resultado	19
4. Conclusión	20
AnexoA: Información Adicional	21
A.4: Tipos de LEDs	21
A.4.0: RGB LEDs	21
A.4.0: LEDs con circuitos integrados	21
A.4.0: RGB LEDs	21

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: TIC is Among Us

Esta primera actividad consiste en diseñar, implementar y probar un sistema de juego colaborativo-competitivo basado en la comunicación entre dispositivos y minijuegos locales, ¡al estilo de Among Us!



El proyecto consiste en la creación de un entorno modular que permita la interacción entre un host central (Raspberry Pi profesor) y varios jugadores conectados (Raspberry Pi alumnos). Cada jugador ejecutará distintos minijuegos físicos o virtuales, generando registros que serán monitoreados por el host para determinar el estado del juego, las puntuaciones y supervivencia de los participantes.

El desarrollo se realizará de forma incremental, de modo que cada ítem se pueda probar de manera local y aislada, culminando con una etapa final de integración y testeo en red.

La estructura del juego se dividirá en cuatro fases principales:

Fase	Descripción
Lobby	Registro inicial de jugadores y verificación de conexión entre dispositivos.
Ronda 1	Primera ronda de minijuegos. Los jugadores obtienen puntos o vida según su desempeño.
Ronda 2	Misma estructura que la ronda 1, pero con penalizaciones y sabotajes introducidos al azar.
Ronda Final	Fase de supervivencia con un temporizador global. Los jugadores pierden vida progresivamente y deben reaccionar rápidamente para mantenerse con vida.

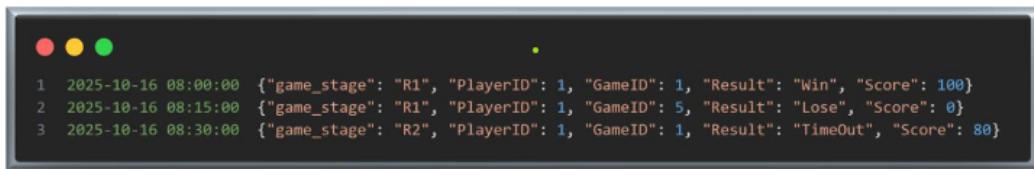
2.1. Ítem 1

El objetivo de este ítem es diseñar los minijuegos que se utilizarán en las rondas 1 y 2, implementando al menos 2 minijuegos por consola y 2 minijuegos basados en sensores físicos con una duración máxima de 15 segundos.

1. El desarrollo de los minijuegos queda a su libre elección, pero los minijuegos basados en sensores deben incluir al menos uno de los siguientes módulos de su kit de 45 Sensores:
2. Sensor DHT11 de temperatura y humedad (KY-015). Documentación de referencia: KY-015 Combi-Sensor (temperature and humidity) - SensorKit
3. Sensor HC-SR04 de distancia por ultrasonido (KY-050). Documentación de referencia: KY-050 Ultrasonic distance sensor - SensorKit
4. Módulo Joystick (KY-023). Documentación de referencia: KY-023 Joystick (eje X-Y) - SensorKit

Nota: Los diagramas de conexión y códigos de prueba se encuentran en sus respectivos Anexos.

Cada minijuego debe generar un logging local de eventos siguiendo un formato unificado que el host central pueda interpretar posteriormente, tomando como base la siguiente estructura:



```
● ● ●
1 2025-10-16 08:00:00 {"game_stage": "R1", "PlayerID": 1, "GameID": 1, "Result": "Win", "Score": 100}
2 2025-10-16 08:15:00 {"game_stage": "R1", "PlayerID": 1, "GameID": 5, "Result": "Lose", "Score": 0}
3 2025-10-16 08:30:00 {"game_stage": "R2", "PlayerID": 1, "GameID": 1, "Result": "TimeOut", "Score": 80}
```

Cada registro del log representa un evento individual de los jugadores, correspondiente a una ejecución o resultado obtenido en un minijuego durante las distintas rondas. Para mayores referencias sobre qué indica cada campo, ver Anexo “Datos de registro”.

2.2. Ítem 2

Implementar un sistema de comunicación entre un jugador y el host central que permita establecer la conexión inicial del juego mediante una estrategia de acuse de recibo (ACK). Este ítem debe implementarse de manera local, simulando tanto el host como el jugador

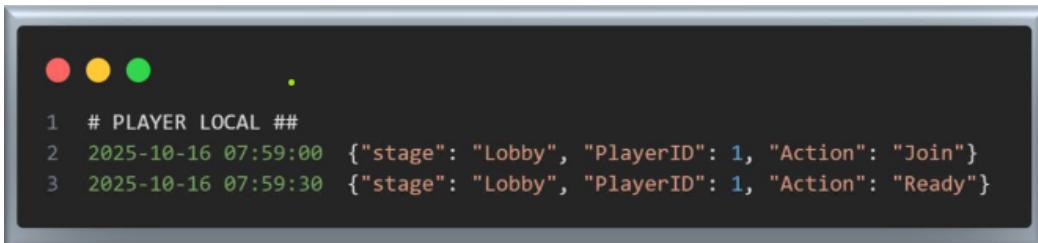
Durante la etapa del Lobby, se darán a lugar las siguientes acciones:

1. Cada jugador deberá generar una entrada en su archivo de logging local de eventos (el mismo utilizado para los minijuegos), incorporando su identificador (“PlayerID”) y un indicador del inicio de conexión (“Action”: “Join”).
2. El archivo de logging deberá ser transferido al host central mediante el protocolo SSH.
3. El host central recibirá el registro del jugador (por ejemplo, “player_events.log”) y emitirá su confirmación de aceptación mediante su propio archivo de logging (“game_status.log”), registrando el evento con el campo “Action”: “Accepted“.
4. Una vez recibida la confirmación de conexión, el jugador deberá actualizar su estado local registrando el evento “Action“: Ready”, indicando que está disponible para iniciar la partida.

Para comprender mejor el flujo de esta operación, a continuación, se muestra el contenido de ejemplo de los logging tanto del host central como del jugador:

Estructura de registro en el host central (game_status.log)

```
1 ## HOST CENTRAL ##
2 2025-10-16 07:59:10 {"stage": "Lobby", "PlayerID": 1, "Action": "Accepted"}
```

Estructura de registro en el jugador (player_events.log)

```
1 # PLAYER LOCAL #
2 2025-10-16 07:59:00 {"stage": "Lobby", "PlayerID": 1, "Action": "Join"}
3 2025-10-16 07:59:30 {"stage": "Lobby", "PlayerID": 1, "Action": "Ready"}
```

2.3. Ítem 3

Integrar los minijuegos desarrollados en el Ítem 1.1 dentro de un ciclo de juego que contemple las etapas Lobby, Ronda 1 y Ronda 2, implementando un sistema de sabotaje pasivo controlado por el host central.

Durante las rondas del juego, el host central tendrá la función de:

- Seleccionar aleatoriamente los minijuegos que cada jugador deberá ejecutar.
- Enviar los registros correspondientes al minijuego activo y, en ocasiones, instrucciones de sabotaje.
- Registrar tanto los resultados de los minijuegos como los sabotajes aplicados en su propio archivo de logging (game_status.log).

El jugador, por su parte, deberá:

- Recibir las indicaciones del minijuego actual desde el host.
- Ejecutar el minijuego asignado y registrar su resultado localmente (player_events.log).
- Aplicar los efectos de sabotaje en caso de ser notificado por el host.

En relación con las rondas, cada una estará compuesta por tres minijuegos elegidos aleatoriamente por el host. El flujo general es el siguiente:

1. **Inicio de ronda:** El campo “stage” de los logs cambia a “R1” o “R2”, según corresponda con la ronda actual.
2. **Selección de minijuego:** el host central escoge aleatoriamente el próximo minijuego, y lo registra en el “game_status.log” asignando un “GameID” y modificando el campo “Action” a “Assign”.
3. **Ejecución:** el jugador lee el registro del host central, ejecuta el minijuego y guarda su resultado.
4. **Sabotaje:** el host central genera un evento de sabotaje de forma aleatoria, y lo registra en el “game_status.log” modificando el campo “Action” a “Sabotage”, y asignando los campos “Effect” y “Value” según el tipo de sabotaje asignado.
5. **Confirmación:** si el jugador lee un registro de sabotaje, aplica el efecto del sabotaje en el minijuego siguiente y lo deja reflejado en su próximo registro local.

Para mayores referencias sobre qué tipos de sabotajes deben aplicarse, ver Anexo “Tipos de Sabotaje”.

Para comprender mejor el flujo de los minijuegos y posibles sabotajes, a continuación, se muestra el contenido de ejemplo de los logging tanto del host central como del jugador: **Estructura de registro en el host central (game_status.log)**

Estructura de registro en el jugador (player_events.log)

2.4. Ítem 4

Desarrollar e integrar la fase final del juego, en la cual el jugador debe mantener su vida por encima de cero durante un periodo de tiempo limitado. Para ello, implementen un minijuego extra con un botón pulsador, el cual enviará registros al host central cada 10 presiones consecutivas. El host central reducirá la vida del jugador progresivamente con el tiempo y la restaurará al detectar los eventos de pulsación.

En relación con las rondas, esta etapa comparte la misma estructura base del sistema de logging utilizado anteriormente, pero adaptada a la dinámica de supervivencia:

1. **Inicio de ronda:** El campo ”stage.en los logs cambia a ”Final”, marcando el inicio de la etapa de supervivencia.
2. **Asignación inicial:** El host central crea el primer registro en su log (“game_status.log”) con la vida inicial del cada jugador (“Life”: 100), y luego inicia un contador con el tiempo de autodes-trucción (por ejemplo, 2 minutos).
3. **Pérdida de vida por tiempo:** En intervalos regulares, el host central actualiza el valor de ”Liferestando unidades según el tiempo transcurrido.
4. **Recuperación de vida por pulsación:** Cada vez que el jugador presione el botón 10 veces, se registra un evento “Action”: Recover.en su log (“player_events.log”). El host central leerá este evento y aumentará la vida en un valor fijo (por ejemplo, +10 unidades).
5. **Finalización:** Si la vida del jugador llega a 0, el host genera un último registro con ”Life”: 0, indicando el término del juego. Si se cumple el tiempo máximo de simulación y un jugador mantiene su vida por sobre 0, se declara ganador. En caso de que varios jugadores sobrevivan la etapa final, se declara ganador al jugador que tenga más vida.

2.5. Solución

2.5.1. Ítem 1

En este proyecto se desarrollaron cuatro mini juegos utilizando una Raspberry Pi: dos utilizan sensores físicos para interactuar con el jugador y los otros dos funcionan únicamente desde la terminal. Estos son:

■ Equation Game

Este minijuego implementa una interfaz gráfica usando *PyQt6*, donde se muestran las instrucciones, la ecuación generada aleatoriamente, un campo para escribir la respuesta, un botón para enviarla, un temporizador visible y una zona donde aparece el puntaje final. Todos estos elementos se organizan verticalmente mediante un **QVBoxLayout** y se estilizan con colores y tamaños de letra para mejorar la claridad visual.

Generación de la Ecuación El juego crea una ecuación lineal aleatoria de la forma $ax + b = c$, donde:

- x es siempre un número entero entre -9 y 9 ,
- a nunca es cero para asegurar que la ecuación tenga solución válida,
- los valores se generan de manera que la solución sea siempre un entero.

El jugador dispone de 15 segundos para ingresar su respuesta y presionar el botón *Submit*. El programa verifica que la entrada sea numérica y luego la compara con la solución correcta.

Sistema de Puntaje El puntaje comienza en 100 puntos. Las deducciones inician únicamente si el jugador tarda más de 5 segundos en contestar. A partir del segundo 6, se descuentan 10 puntos por cada segundo adicional, hasta un mínimo de 0 puntos. Si el jugador responde rápido (antes de 6 segundos), mantiene los 100 puntos intactos.

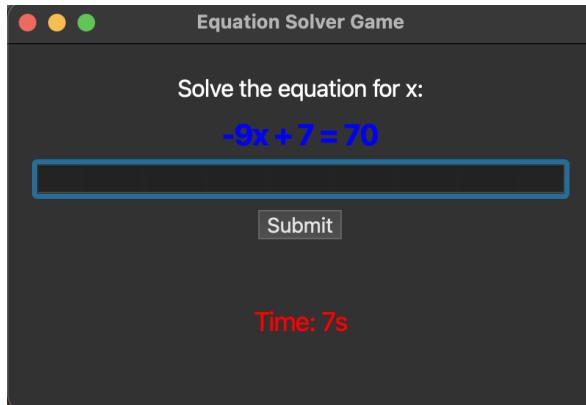
Finalización del Juego El juego puede terminar de dos maneras:

- si el jugador ingresa la respuesta correcta, o
- si el temporizador llega a cero.

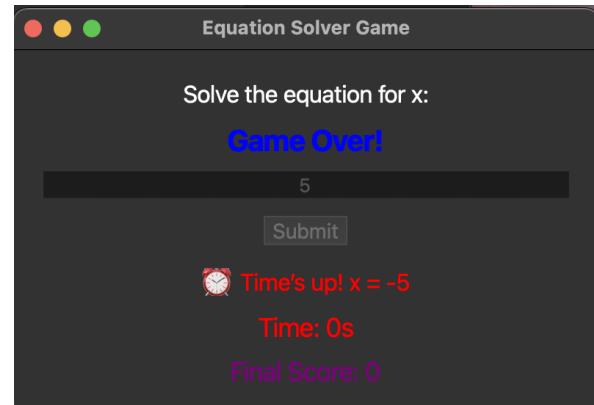
En cualquiera de estos casos:

- se desactivan el cuadro de respuesta y el botón,
- se muestra la solución correcta,
- se detiene el temporizador,
- se presenta el puntaje final,
- y la ventana se cierra automáticamente después de dos segundos.

De esta forma, el resultado final queda disponible para el módulo principal que ejecutó el mini-juego.



(a) Inicio del minijuego



(b) Final del minijuego

■ Memory Game

Este minijuego consiste en un juego de memoria implementado con *PyQt6*. La interfaz gráfica incluye un temporizador visible, un espacio donde se muestra la secuencia generada, un campo de texto para escribir la respuesta y dos botones: uno para iniciar la partida y otro para enviar la secuencia recordada. Todos los elementos se organizan verticalmente mediante un *QVBoxLayout* y se estilizan para mejorar la claridad visual.

Funcionamiento General

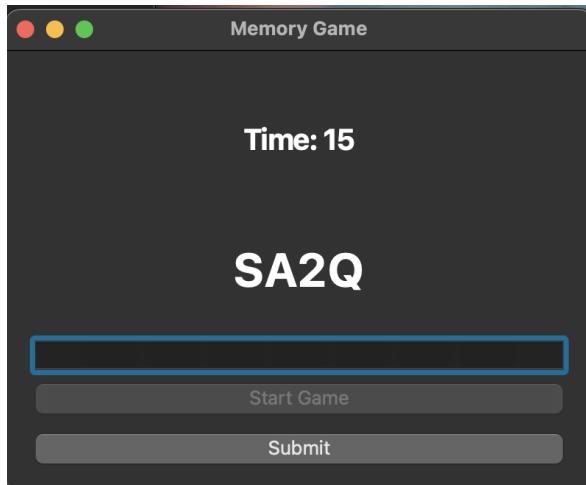
El jugador dispone de un tiempo total de 15 segundos para intentar superar hasta 5 rondas. La primera ronda comienza con una secuencia aleatoria de 4 caracteres compuestos por letras y números. En cada ronda siguiente, la longitud de la secuencia aumenta en un carácter, elevando la dificultad del juego. La secuencia generada se muestra durante un segundo y luego se oculta automáticamente. En ese momento, el jugador debe escribir exactamente la misma secuencia en el cuadro de texto y presionar el botón correspondiente para validarla.

Sistema de Puntaje

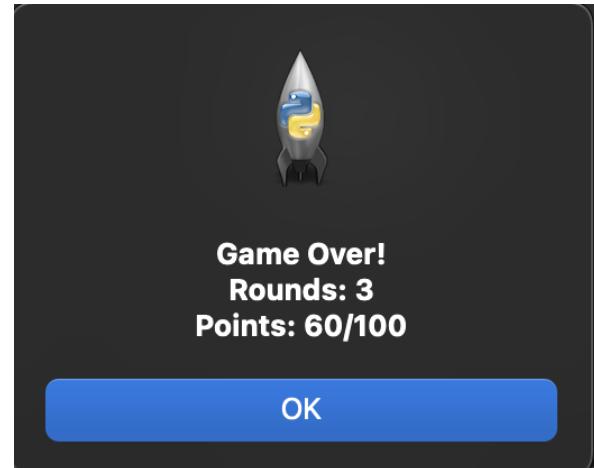
El sistema de puntos es directo:

- cada ronda superada otorga 20 puntos,
- el puntaje máximo posible es 100 puntos,
- si el jugador ingresa una secuencia incorrecta, la partida termina inmediatamente mostrando un mensaje de *Game Over*.

El juego también finaliza si el temporizador global llega a cero. En ese caso, se registran los puntos acumulados hasta ese momento.



(a) Inicio del minijuego



(b) Final del minijuego

■ Balloon game

Este minijuego consiste en controlar un cursor en pantalla mediante un joystick analógico y reventar globos usando un botón físico conectado a una Raspberry Pi. El objetivo principal es evitar que alguno de los globos llegue a la parte inferior de la pantalla antes de ser reventado.

Hardware Utilizado

- **ADS1115 (ADC) para el joystick:** Se leen los ejes X y Y desde los canales P0 y P1, obteniendo valores entre 0 y 65535 que se normalizan al rango 0–1. Estos valores se traducen en movimiento horizontal y vertical del cursor.
- **Botón físico:** Un botón conectado al pin GPIO,17 permite “reventar” globos al presionarlo. Cada pulsación verifica si el cursor está intersectando un globo para eliminarlo.

Reglas del Juego

- **Movimiento:** El jugador controla un cuadrado blanco de 40,px utilizando el joystick analógico.
- **Reventar globos:** Al presionar el botón físico se revisan colisiones entre el cursor y los globos. Los globos que colisionan desaparecen.
- **Derrota:** El jugador pierde inmediatamente si un globo alcanza la parte inferior de la ventana (posición $y > 800$).
- **Victoria:** El juego tiene una duración total de 10,segundos. Al finalizar, si no queda ningún globo en pantalla, el jugador gana; de lo contrario, pierde.

Curva de Dificultad

Los globos aparecen inicialmente cada 800,ms. Cada vez que surge un nuevo globo, este intervalo se reduce en 20,ms hasta llegar a un mínimo de 250,ms. Esto genera un aumento progresivo de la dificultad, ya que con el tiempo los globos aparecen con mayor frecuencia. **Interfaz Gráfica (PyQt6)** La interfaz se dibuja utilizando **QPainter**, permitiendo animaciones fluidas a 60,FPS. Los elementos visuales son:

- **Fondo oscuro** para resaltar los globos.
- **Globos** de 40,px de colores aleatorios (rojo, azul, amarillo, verde, magenta, cian), cada uno con una velocidad de caída aleatoria.
- **Cursor del jugador** representado como un cuadrado blanco de 40,px.
- **Mensaje de fin de juego** que muestra "WIN" o "LOSE" centrado en pantalla.

Flujo del Juego

- Al iniciar, se activan tres temporizadores: uno para actualizar la animación a 60,FPS, otro para generar globos periódicamente y un temporizador global de 10,segundos.
- En cada actualización se leen los valores del joystick, se actualiza el cursor, los globos descienden y se verifican colisiones si el botón está presionado.
- Si un globo escapa por la parte inferior, el juego termina en derrota.
- Cuando se cumplen los 10,segundos, se determina si el jugador ganó o perdió según la cantidad de globos restantes.

Función Pública El minijuego se ejecuta mediante la función:

```
def run_balloon_game():
```

La cual bloquea la ejecución hasta que se cierre la ventana y devuelve el valor "Win" o "Lose" para ser usado por el programa principal.

■ Reaction Game

Este minijuego consiste en medir la rapidez del jugador al presionar el botón correcto según el color indicado en pantalla. Está desarrollado con *PyQt6* y puede utilizar botones físicos conectados a una Raspberry Pi o, si no hay hardware disponible, el teclado como alternativa. La interfaz gráfica muestra instrucciones, el color que el jugador debe presionar y un contador de puntaje actualizado después de cada ronda.

Hardware Utilizado

- **Botones físicos (GPIO):** Si la Raspberry Pi tiene acceso a los pines GPIO, se utiliza un botón negro (GPIO 27), uno azul (GPIO 18) y uno rojo (GPIO 17). Cada uno representa una respuesta válida según el color solicitado.
- **Entrada alternativa por teclado:** En dispositivos sin GPIO, las teclas **1**, **2** y **3** reemplazan a los botones negro, azul y rojo respectivamente.

Funcionamiento General El juego consta de 3 rondas. En cada una, el programa espera entre 2 y 5 segundos y luego muestra aleatoriamente uno de los colores: *Black*, *Blue* o *Red*. Inmediatamente comienza a medir el tiempo de reacción del jugador desde que se muestra el color hasta que

presiona el botón correspondiente. Cada ronda registra tanto el color elegido como el tiempo exacto de reacción del jugador.

Sistema de Puntaje El puntaje máximo total es 100 puntos, distribuidos entre las 3 rondas (33, 33 y 34 puntos). El puntaje de cada ronda se determina de la siguiente forma:

- **Si el color presionado es incorrecto:** la ronda otorga 0 puntos.
- **Si el color es correcto y el tiempo de reacción es menor o igual a 0.5 segundos:** se otorga el puntaje completo de la ronda.
- **Si el tiempo supera los 0.5 segundos:** cada 0.1 segundos adicionales se descuentan 3 puntos.

El puntaje mínimo por ronda es 0. Al finalizar las tres rondas, el juego suma los puntajes obtenidos y los muestra en pantalla junto con un detalle del tiempo de reacción de cada intento.

Interfaz Gráfica (PyQt6) La interfaz utiliza un QVBoxLayout para organizar los elementos principales:

- un texto con el color que debe presionar el jugador;
- un área de resultados donde se muestran los tiempos y puntos de cada ronda;
- un contador de puntaje total actualizado en tiempo real.

La ventana se mantiene simple para resaltar la instrucción principal y facilitar la reacción rápida del usuario.

Flujo del Juego

- Tras un retraso inicial de 1 segundo, se inicia la primera ronda.
- En cada ronda se genera un color al azar y se registra el tiempo de inicio.
- Al presionar un botón físico o una tecla, se calcula el tiempo de reacción y se asigna el puntaje.
- Si aún quedan rondas, se espera entre 2 y 5 segundos antes de mostrar el siguiente color.
- Al completar las 3 rondas, se muestra el resumen de resultados y el puntaje final.

Función Pública El minijuego se ejecuta mediante:

```
def run_reaction_game():
```

La función abre la ventana, espera a que termine el juego y retorna un diccionario con el resultado y el puntaje total obtenido.

Finalmente, la siguiente imagen muestra un esquema de los circuito realizado para los minijuegos:

2.5.2. Ítem 2

Para esta sección se realizó únicamente el código del jugador. Este código implementa la lógica del jugador dentro del sistema de comunicación con el host central, siguiendo los pasos solicitados en el enunciado del ítem 2. El flujo se basa en el uso de archivos de logging en formato JSON y en la transferencia remota mediante SSH. Todo el comportamiento está encapsulado dentro del archivo principal del jugador, el cual administra la conexión inicial, la espera del acuse de recibo (ACK) y la ejecución de los minijuegos. Su funcionamiento es el siguiente:

1. Estructura general del programa

El código del jugador está organizado en varios componentes principales que trabajan de manera coordinada para establecer la comunicación con el host central, registrar eventos y ejecutar las rondas del juego. Para ello se emplean librerías estándar de Python como `os`, `json` y `time`, además de la librería `paramiko`, que permite conectarse al host mediante SSH y transferir archivos utilizando el protocolo SFTP.

```

1 import os, json, time, random
2 from datetime import datetime
3 import paramiko
4
5 # --- Imported minigames ---
6 from minigames.memory_game import play_memory_game
7 from minigames.juego2_ecuacion import run_equation_game
8 from minigames.juego1_colores import run_reaction_game
9 from minigames.juego_globos import run_balloon_game
0

```

Fig. 3: Librerías utilizadas para la actividad

2. Registro inicial de conexión (“Join”).

Al comenzar la ejecución, el jugador crea el archivo local `player_events.log` dentro de la carpeta `logs/`. Allí escribe la primera entrada correspondiente a la etapa de Lobby, registrando su identificador (“PlayerID”) y la acción “Join”. Esta entrada se almacena en formato JSON, garantizando compatibilidad con el sistema de lectura del host.

```

def log_player_event(stage, game_name, action, result):
    entry = {
        "timestamp": datetime.now().isoformat(),
        "stage": stage,
        "PlayerID": PLAYER_ID,
        "GameName": game_name,
        "Action": action,
        "Result": result
    }
    (constant) PLAYER_LOG: LiteralString
    with open(PLAYER_LOG, "a", encoding="utf-8") as f:
        f.write(json.dumps(entry, ensure_ascii=False) + "\n")
    upload_log_via_ssh()

```

Fig. 4: Función para crear el contenido del .log

3. Transferencia del archivo de logging al host.

Después de registrar cada evento, el archivo de logging se envía automáticamente al host central utilizando el protocolo SSH. Para ello, el programa establece una conexión empleando la librería `paramiko`, abre un canal SFTP y transfiere el archivo al directorio remoto definido por el profesor. Si la carpeta no existe, el jugador intenta crearla antes de efectuar la carga. Con esto, el host recibe constantemente los eventos del jugador conforme ocurren.

```
def upload_log_via_ssh():
    try:
        ssh = paramiko.SSHClient()
        ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
        ssh.connect(HOST_IP, HOST_PORT, HOST_USERNAME, HOST_PASSWORD)

        sftp = ssh.open_sftp()

        # Ensure directory exists
        try:
            sftp.stat(HOST_TARGET_FOLDER)
        except:
            sftp.mkdir(HOST_TARGET_FOLDER)

        # Upload log
        remote_path = HOST_TARGET_FOLDER
        sftp.put(PLAYER_LOG, remote_path)

        sftp.close()
        ssh.close()
        print(f"[SSH] Log file uploaded → {remote_path}")

    except Exception as e:
        print(f"[SSH ERROR]: {e}")
```

Fig. 5: Función para crear la conexión entre el jugador y el host

4. Recepción del acuse de recibo desde el host (“Accepted”).

Para recibir la confirmación del host, el jugador revisa periódicamente el archivo remoto `game_status.log`. Cada segundo, el programa abre dicho archivo mediante SSH, lee la última línea y la interpreta como un objeto JSON. Cuando detecta un evento cuyo campo `Action` es “Accepted”, el jugador reconoce que el host ha confirmado su incorporación a la partida. Este mecanismo implementa directamente la estrategia de acuse de recibo (ACK) requerida en el enunciado.

```
def wait_for_host_accepted():
    STATUS_FILE = "/home/minipc/Desktop/Game_App/game_status.log"

    print("Waiting for host... (looking for Action: Accepted)")

    last_line = None

    while True:
        try:
            ssh = paramiko.SSHClient()
            ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
            ssh.connect(HOST_IP, HOST_PORT, HOST_USERNAME, HOST_PASSWORD)

            sftp = ssh.open_sftp()

            try:
                remote_file = sftp.open(STATUS_FILE, "r")
                lines = remote_file.readlines()

                if lines:
                    new_line = lines[-1].strip()

                    if new_line != last_line:
                        last_line = new_line
                        print(f"[HOST MESSAGE] {new_line}")

                    try:
                        data = json.loads(new_line)
                    except:
                        print("Invalid JSON, skipping")
                        continue

                    if data.get("Action") == "Accepted":
                        print("Host accepted. Beginning rounds!")
                        remote_file.close()
                        sftp.close()
                        ssh.close()
                        return # DONE - return control to play_game()

            finally:
                remote_file.close()

        except IOError:
            pass # File not created yet

        sftp.close()
        ssh.close()

    except Exception as e:
        print(f"Error while reading host file: {e}")

    time.sleep(1)
```

Fig. 6: Función para esperar la respuesta del host

5. Actualización del estado a “Ready”.

Tras recibir el mensaje de aceptación, el jugador continúa con el flujo normal del juego registrando sus estados durante cada ronda. Aunque en esta versión el mensaje explícito “Ready” no se genera inmediatamente después del ACK, el sistema mantiene el mismo formato de registro, incorporando en cada etapa la acción realizada, el identificador del jugador y el resultado asociado. Cada uno de estos eventos se sincroniza nuevamente con el host mediante SSH.

6. Ejecución de las rondas y de los minijuegos.

Luego de ser aceptado por el host, el jugador ejecuta tres rondas consecutivas. En cada una de ellas selecciona aleatoriamente uno de los cuatro minijuegos disponibles: *MemoryGame*, *EquationGame*, *ReactionGame* o *BalloonGame*. Antes de iniciar cada minijuego, el programa registra un evento de tipo “Start”, y al finalizarlo registra un evento “End” indicando si el jugador obtuvo un resultado de “Win” o “Lose”. Como en las etapas anteriores, cada evento queda almacenado en el archivo local y es enviado inmediatamente al host central.

```
def play_game():
    print("Sending JOIN request to host...")
    log_player_event("Connection", "None", "Join", "Sent")

    wait_for_host_accept()

    ROUNDS = 3

    for round_num in range(1, ROUNDS + 1):
        print(f"\n===== ROUND {round_num}/3 =====")

        # Random game selection
        game = random.choice([
            "MemoryGame",
            "EquationGame",
            "ReactionGame",
            "BalloonGame"
        ])

        log_player_event(f"Round{round_num}", game, "Start", "Running")
        print(f"Starting game - {game}")


```

```
# Run selected game
if game == "MemoryGame":
    result = play_memory_game()

elif game == "EquationGame":
    score = run_equation_game()
    result = "Win" if score > 0 else "Lose"

elif game == "ReactionGame":
    win = run_reaction_game()
    result = "Win" if win else "Lose"

elif game == "BalloonGame":
    win = run_balloon_game()
    result = "Win" if win else "Lose"

else:
    result = "Lose"

log_player_event(f"Round{round_num}", game, "End", result)
print(f"[RESULT] {game} finished - {result}")

time.sleep(1)

print("\nAll 3 rounds completed!")

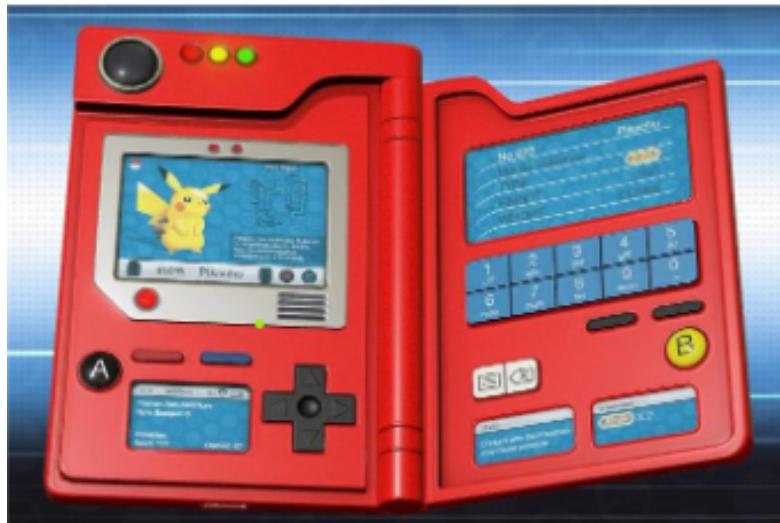

```

Fig. 7: Función principal

2.5.3. Item 3

3. Actividad n°2: Programando una Pokédex

Esta segunda actividad consiste en diseñar e implementar una interfaz gráfica interactiva que simule una Pokédex: un dispositivo inteligente capaz de mostrar información sobre distintos Pokémon.



3.1. Ítem 1

Para empezar a crear su Pokédex, su primer objetivo será diseñar una interfaz gráfica en Raspberry Pi que sea capaz de interactuar con un módulo LED (KY-009, KY-011, KY-016 o KY- 029) y los siguientes widgets de PyQt:

- QPushButton - QPushButton Class — Qt Widgets 6.5.3
- QLabel - QPushButton Class — Qt Widgets 6.5.3
- QGroupBox - QGroupBox Class — Qt Widgets — Qt 6.10.0

Luego, construyan su interfaz gráfica inspirándose en el diseño original de la Pokédex, considerando los siguientes puntos para darle funcionalidad desde un código Python:

1. Definir la estructura de datos que contendrá la información de los Pokémon (al menos 10 diferentes). Cada registro deberá ser un archivo tipo .json o .txt que contenga al menos los siguientes datos:

```
● ● ●  
1 {  
2   "nombre": "Charmander",  
3   "tipo": "fuego",  
4   "descripcion": "Un lagarto que respira fuego.",  
5   "imagen": "images/charmander.png"  
6 }
```

2. Uno QLabel deberá usarse para mostrar la imagen del Pokémon actual, la cual debe cargarse con ayuda del método setPixmap().
3. Otro 2 QLabel's deberán mostrar la información del tipo y descripción del Pokémon actual, usando para ello el método setText().
4. Los QPushButtons deberán cambiar dinámicamente la información mostrada en la interfaz respecto del Pokémon actual. El primero será para avanzar al Pokémon siguiente, el segundo será para retroceder al anterior, y el último será para elegir un Pokémon aleatorio.
5. Programen el LED para que cambie de color según el tipo de Pokémon actual (por ejemplo, color azul para un Pokémon tipo agua, color rojo para un Pokémon tipo fuego, etc.)

3.2. Solución

3.2.1. Diseño

El diseño principal de la Pokédex fue realizado usando Qt Designer. Para su construcción se utilizaron los siguientes elementos:

- 4 QPushButton
- 8 QLabel
- 1 QStatusBar

De estos elementos, dos QLabel están siendo utilizados como imágenes, y uno de ellos corresponde a un LED virtual que cambia de color dependiendo del tipo del Pokémon. El diseño final en Qt Designer puede ser visualizado en la imagen 8.



Fig. 8: Diseño en Qt Designer

3.3. Funcionamiento

Una vez importado el diseño a python, se crearon tres diferentes códigos llamados `main.py`, `pokedex_ui.py` y `pokedex.py`. estos códigos funcionan de la siguiente forma:

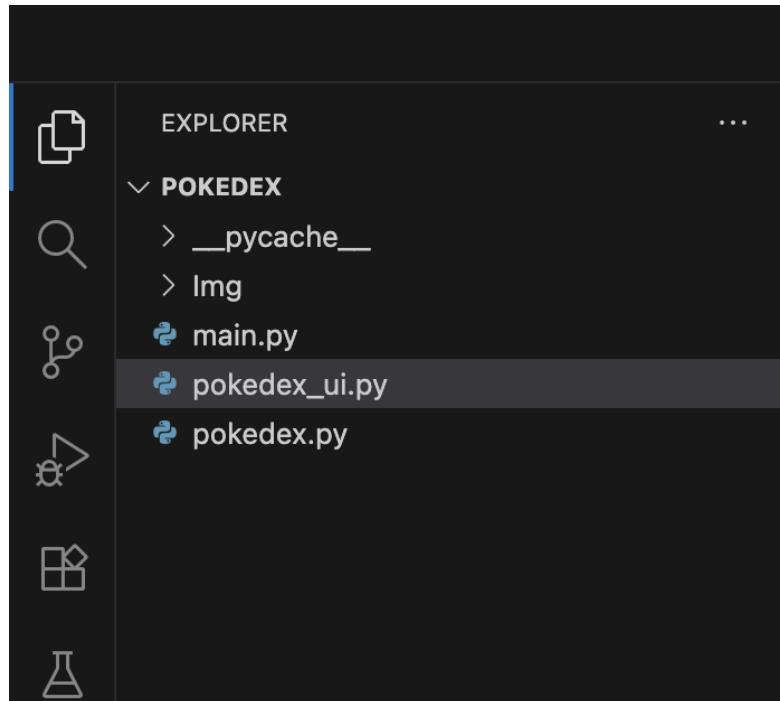


Fig. 9: Distribución de los códigos

▪ **pokedex_ui.py**

Código sujeto al diseño realizado en Qt Designer, es el resultado de transformar el diseño gráfico en formato python. Esta pestaña será utilizada como la página principal de la interfaz.

▪ **pokedex.py**

Código encargado de asignar funciones a los elementos utilizados en el diseño para modificar la pokedex y almacenar las características de cada pokemon. Dichos elementos interactúan de la siguiente manera:

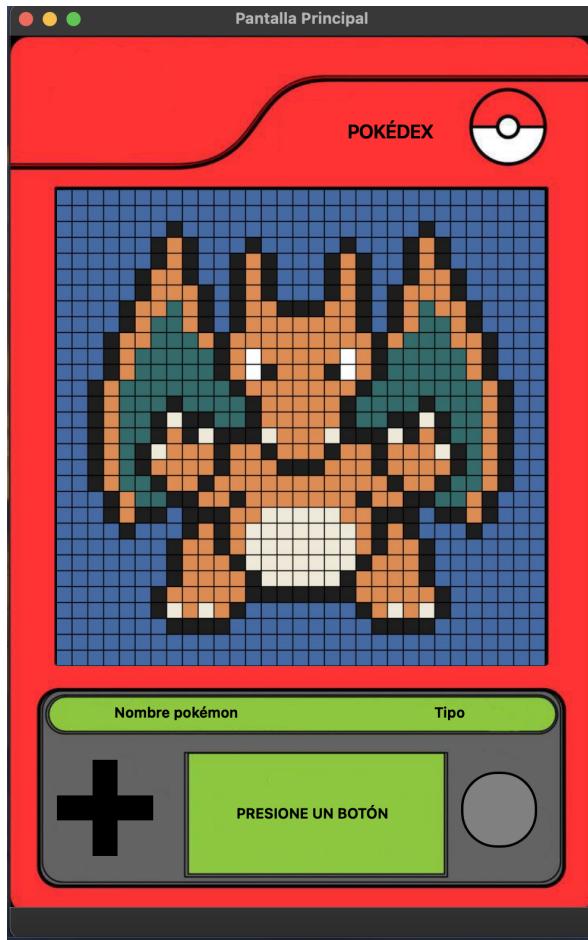
- Boton superior: Elije un pokemon al azar
- Boton inferior: Regresa a la página principales
- Boton izquierdo: Vuelve al pokemon anterior
- Boton derecho: Avanza al siguiente pokemon
- LED: Circulo en la esquina inferior derecha de la pokedex. Cambia de color dependiendo del tipo del pokemon.

▪ **main.py**

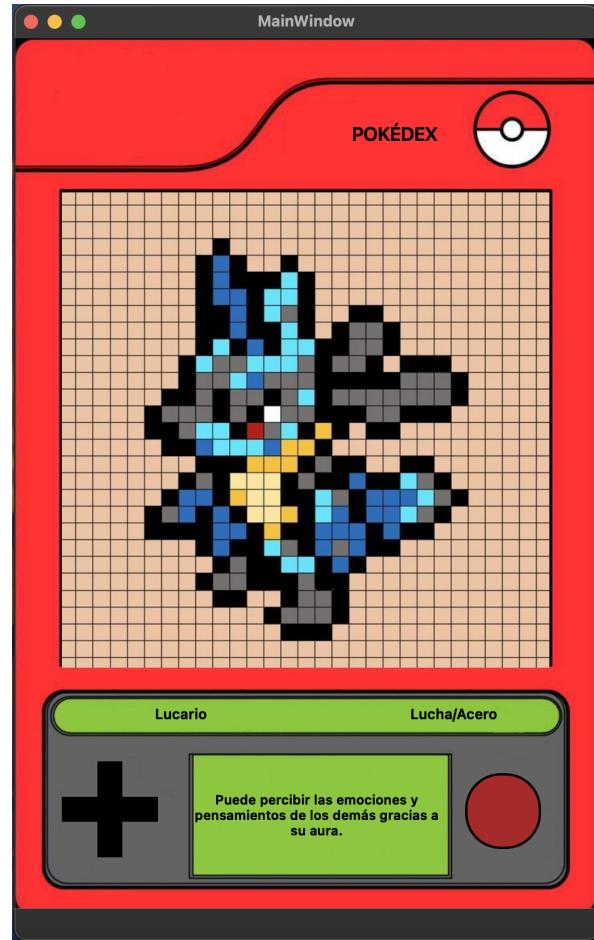
Código principal de la interfaz. Este es el código que debe ejecutarse para iniciar la aplicación. Al hacerlo, la interfaz abre la página principal. Para cambiar a la Pokédex, se debe presionar cualquiera de los cuatro botones disponibles

3.4. Resultado

Finalmente, la interfaz gráfica final puede observarse en la imagen 9.



(a) Pantalla principal



(b) Pokédex

4. 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.

Anexo A: Información Adicional

A.4: Tipos de LEDs

A.4.0: 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. 11: LED de cátodo transparente común RGB. Fuente:

A.4.0: 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. 12: Primer plano del LED de ciclo lento de 5 mm. Fuente:

A.4.0: 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.

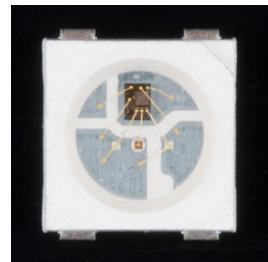


Fig. 13: Primer plano direccional WS2812B. Fuente: