Departamento de Ingeniería Eléctrica
Facultad de Ingeniería
Universidad de Concepción
Concepción, Chile.

549253-1 Taller de Aplicación TIC I

Informe MiniProyecto 2

Profesor: Vincenzo Caro Fuentes

Integrantes: Joaquin Mardones Gallegos

Oscar Ortiz Molina

23 de noviembre de 2025

Concepción, 23 de noviembre de 2025

Resumen

En este informe presentamos el trabajo realizado para el MiniProyecto 2, donde desarrollamos un sistema de juegos interactivos usando la Raspberry Pi 5. El proyecto se dividió en dos grandes partes: primero, creamos una serie de minijuegos bajo la dinámica de "Among Us", pero **personalizados mayoritariamente con una temática de Pokémon**, mezclando pruebas físicas con sensores y juegos virtuales; y segundo, programamos una Pokédex funcional que reconoce tipos de Pokémon y enciende luces LED según corresponda. Lo más importante fue lograr que todo funcionara al mismo tiempo sin pegarse, usando hilos de ejecución, y que todos los juegos guardaran sus resultados en un registro común tal como se pedía.

Índice

1. Introducción

Para este segundo MiniProyecto del curso Taller de Aplicación TIC I, nos propusimos llevar al límite las capacidades de la Raspberry Pi 5, no solo como un computador pequeño, sino como el cerebro de un sistema interactivo completo. La temática central fue "TIC is Among Us", lo que implicaba crear minijuegos que simularan las tareas de mantenimiento de una nave. Nosotros decidimos adaptar estas tareas utilizando una estética de **Pokémon** para la mayoría de los desafíos, dándole un toque personal al proyecto.

Nuestro objetivo fue diseñar cuatro minijuegos distintos: dos que obligaran al jugador a interactuar con el mundo real usando sensores y botones, y dos que fueran puramente de software. Además, teníamos el desafío de que todos estos juegos generaran reportes (logs) automáticos para que, en una situación real, un servidor central pudiera saber qué está haciendo cada jugador.

Para lograr esto, programamos todo en Python, usando librerías como `PyQt` para las interfaces gráficas y `gpiozero` para controlar el hardware. Uno de los retos más grandes que enfrentamos fue hacer que la interfaz gráfica no se congelara mientras la Raspberry Pi leía los sensores, lo que nos obligó a aprender y aplicar programación con hilos (threads).

En este informe detallaremos cómo conectamos cada componente, cómo resolvimos los problemas de programación y cómo logramos integrar todo en un sistema funcional.

2. Configuración del Entorno de Desarrollo

Antes de abordar los minijuegos, establecimos el flujo de trabajo para asegurar la reproducibilidad del proyecto en la Raspberry Pi 5.

2.1. Herramientas y Dependencias

A diferencia de iteraciones anteriores, configuramos el entorno de desarrollo directamente sobre el hardware ("on-device development") para agilizar las pruebas con los sensores GPIO.

- **Sistema Operativo:** Raspberry Pi OS (64-bit).
- **IDE:** Visual Studio Code instalado localmente en la Raspberry Pi. Esto permitió editar y depurar el código en tiempo real.
- **Librerías Principales:**
  - `PyQt5` y `PyQt6`: Para las interfaces gráficas de usuario.
  - `gpiozero`: Para el control abstracto de sensores y actuadores.
  - `logging` y `json`: Para la generación de registros estandarizados.

3. Actividad 1.1: Minijuego Físico “Reflejos Supersónicos”

Este módulo utiliza un sensor de distancia para medir el tiempo de reacción del usuario con precisión de milisegundos, cumpliendo con el requisito de integración de sensores físicos.

3.1. Dinámica y Funcionamiento

El objetivo del juego es poner a prueba los reflejos del jugador ante un estímulo sorpresivo.

1. **Inicio:** El juego comienza con una luz verde indicando que el sensor está listo.
2. **Espera:** El sistema entra en un tiempo de espera aleatorio (entre 2 y 5 segundos) para evitar que el jugador prediga la señal.
3. **Estímulo:** De repente, se enciende una luz roja y suena el buzzer. En ese instante comienza a correr el cronómetro interno.
4. **Reacción:** El jugador debe colocar su mano frente al sensor ultrasónico (a menos de 10 cm) lo más rápido posible.
5. **Resultado:** Si el tiempo de reacción es menor a 450ms, se considera una victoria. El sistema muestra el tiempo exacto en pantalla.

3.2. Montaje del Circuito y Protecciones

Utilizamos el sensor ultrasónico **HC-SR04**. Un desafío técnico importante es que el pin *Echo* de este sensor emite señales lógicas a 5V, mientras que los pines GPIO de la Raspberry Pi 5 operan estrictamente a 3.3V.

Solución Implementada: Diseñamos un divisor de voltaje utilizando dos resistencias ($R_1 = 1k\Omega$ y $R_2 = 2k\Omega$). Aplicando la ley de Ohm, reducimos el voltaje de retorno a un nivel seguro ($\approx 3.3V$).

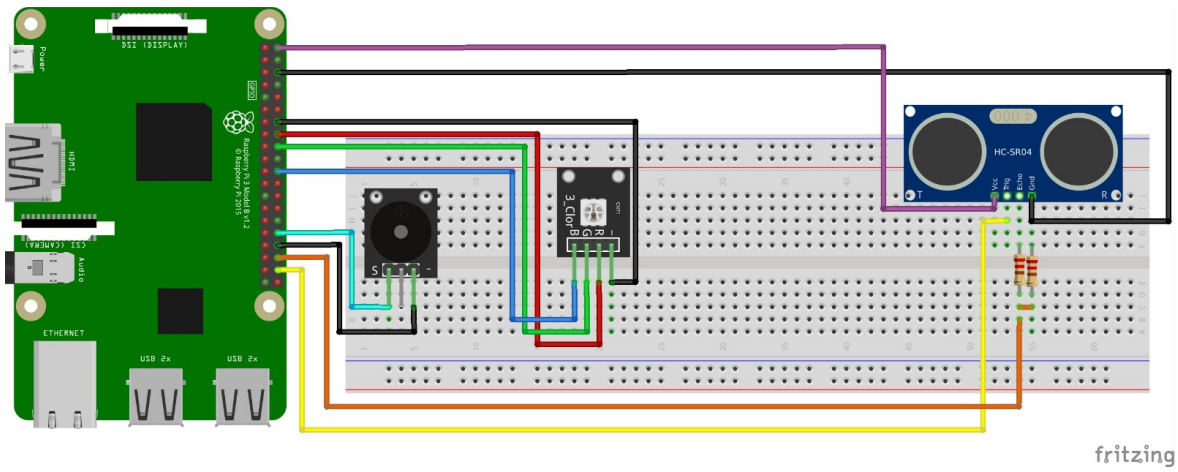

Fig. 1: Esquemático del juego Reflejos. Se destaca el divisor de voltaje en la conexión del pin Echo (GPIO 24).

3.3. Lógica del Código (Multithreading)

El desafío de software fue leer el sensor continuamente sin bloquear la interfaz gráfica. Para resolver esto, implementamos la clase `GameWorker` que hereda de `QObject`. Esta clase ejecuta la lógica de espera aleatoria y el *polling* del sensor en un hilo secundario (*Thread*), liberando al hilo principal para renderizar la GUI fluidamente.

```
# Logica en el hilo secundario (GameWorker)
self.instruction_signal.emit("YA!!!", "red")
buzzer.play(440)
tiempo_inicio = time()

# Polling activo: Se ejecuta sin bloquear la GUI principal
while sensor.distance * 100 > DISTANCIA_ACTIVACION_CM:
    if not self.running: return
    sleep(0.001) # Pausa minima para liberar CPU

tiempo_fin = time()
tiempo_reaccion = (tiempo_fin - tiempo_inicio) * 1000
```

4. Actividad 1.2: Minijuego Físico “Simón Dice”

Este juego recrea el clásico desafío de memoria visual y auditiva, integrando una interfaz gráfica con controles de hardware externos.

4.1. Dinámica y Funcionamiento

El juego consiste en memorizar y repetir secuencias de colores. Es fundamental destacar la **flexibilidad de la interacción**:

1. **Visualización:** El jugador observa la pantalla para ver la secuencia que debe memorizar (los botones virtuales se iluminan).
2. **Ejecución Híbrida:** Aunque el juego fue diseñado para jugarse presionando los **pulsadores físicos** en la protoboard, el sistema también admite hacer clic con el mouse sobre los botones en la pantalla. Ambos métodos son válidos y se registran de la misma manera.
3. **Progresión:** Si la secuencia ingresada (física o virtual) es correcta, el nivel aumenta; de lo contrario, el juego termina.

4.2. Hardware e Interfaz

El circuito consta de 4 botones pulsadores conectados a los pines GPIO 17, 27, 22 y 10. Se configuraron con resistencias *Pull-Up* internas. Además, se integró un **buzzer pasivo** (GPIO 16) para generar la retroalimentación sonora.

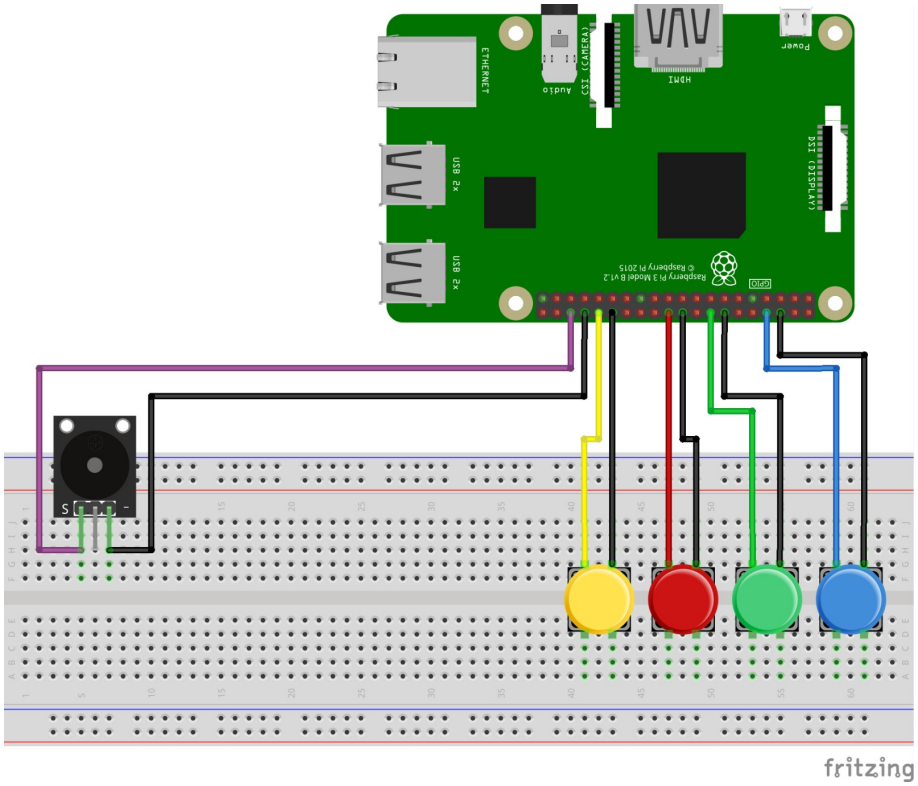

Fig. 2: Conexión de 4 botones y buzzer. La retroalimentación visual ocurre en la pantalla y la física mediante el sonido.

4.3. Sincronización y Manejo de Errores

La interfaz gráfica (GUI) se diseñó para funcionar como espejo de la acción física: los indicadores en pantalla se iluminan tanto cuando la CPU dicta la secuencia como cuando el usuario presiona los botones reales o hace clic en la GUI, ofreciendo feedback inmediato.

El código incluye bloques `try-except` robustos para manejar la reproducción de sonido y un método de limpieza (`cleanup`) para liberar los recursos GPIO al terminar.

```
@pyqtSlot(str)
def process_player_input(self, color):
    # Feedback inmediato (sonido y luz)
    self.play_color(color, duration=0.2)

    # Verificacion de la secuencia
    if color == self.sequence[self.player_index]:
        self.player_index += 1
        if self.player_index == len(self.sequence):
            self.status_signal.emit("Correcto!")
            self.computer_turn() # Siguiente nivel
    else:
        self.game_over() # Error
```

Fig. 3: Interfaz gráfica del Simón Dice. Los botones en pantalla son interactivos, permitiendo jugar con el mouse además del hardware.

5. Actividad 1.3: Minijuego Virtual "Flappy Pokémon"

Cumpliendo con el requisito de software puro, desarrollamos una versión personalizada del "Flappy Bird" utilizando temática de Pokémon.

5.1. Dinámica y Funcionamiento

El juego es un *endless runner* (corredor infinito) donde la precisión es clave.

- **Objetivo:** Guiar a un Pokémon volador (como **Pidgeot**) a través de aperturas en tuberías verticales verdes que se mueven hacia la izquierda.
- **Controles:** El personaje cae constantemente debido a una gravedad simulada. El jugador debe presionar la **Barra Espaciadora** para "aletear" e impulsarse hacia arriba.
- **Puntaje:** Cada par de tuberías superado suma un punto. El juego termina si el personaje toca el suelo, el techo o una tubería.

5.2. Diseño de Software (PyQt6)

Para este juego, actualizamos el stack tecnológico a **PyQt6**. Esto nos permitió utilizar el módulo `QGraphicsScene`, que ofrece un rendimiento superior para manejar sprites 2D y detección de colisiones en tiempo real comparado con los widgets tradicionales.

Los recursos gráficos utilizados (sprites del Pokémon y las tuberías) fueron obtenidos de repositorios de libre acceso en internet y se cargan dinámicamente desde una carpeta local de *assets*.

```
def update_physics(self):
    # Simulacion de gravedad constante
    self.velocity_y += GRAVITY
    self.setY(self.y() + self.velocity_y)

def keyPressEvent(self, event):
    # Deteccion de tecla Espacio
    if event.key() == Qt.Key.Key_Space:
        self.bird.flap() # Impulso negativo (hacia arriba)
```

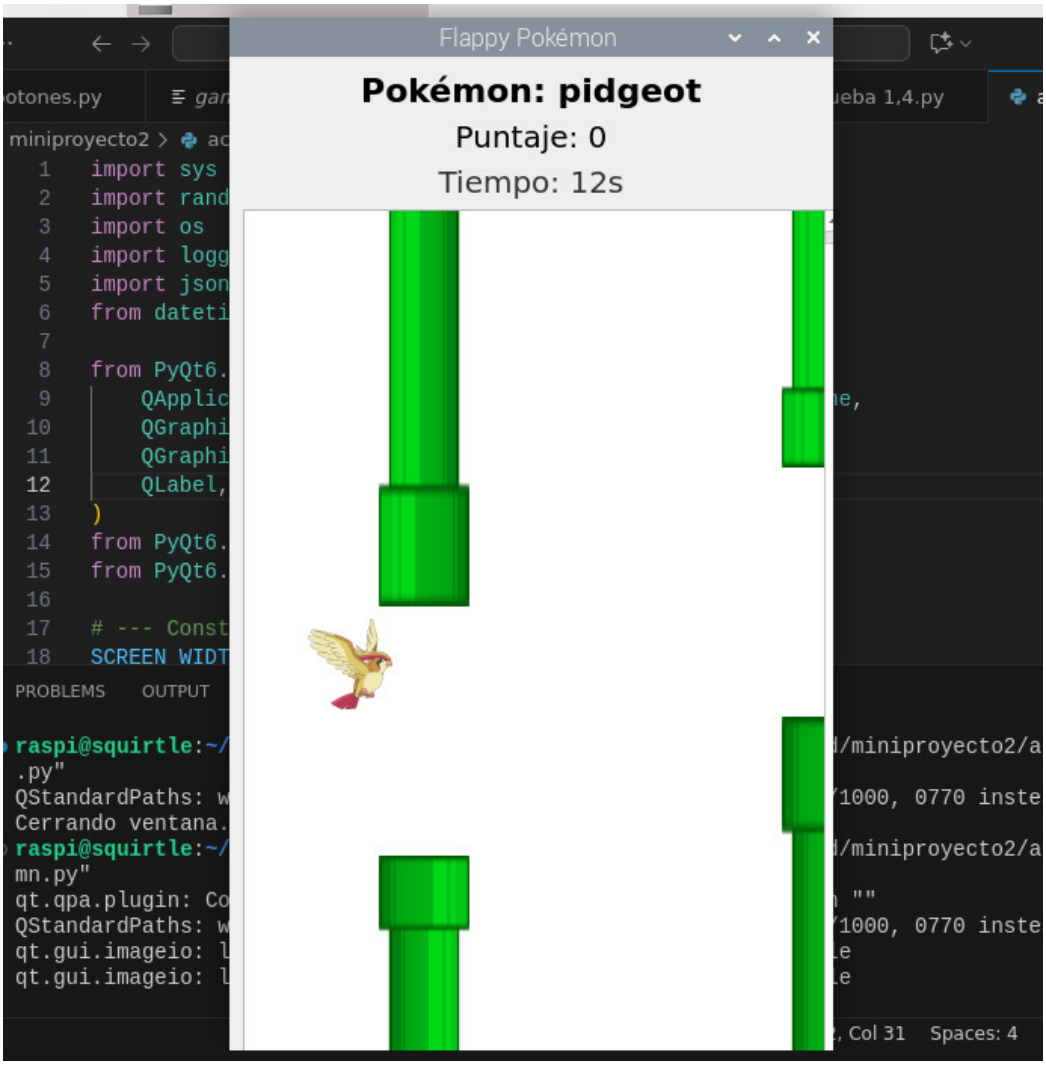

Fig. 4: Captura del juego Flappy Pokémon. Se observa el sprite de **Pidgeot** navegando entre las tuberías, con el contador de tiempo y puntaje en la parte superior.

6. Actividad 1.4: Minijuego Virtual "Lluvia de Pokémon"

El cuarto módulo es un juego tipo "Arcade" de recolección, diseñado para probar la coordinación del usuario bajo presión de tiempo.

6.1. Dinámica y Funcionamiento

El jugador controla al entrenador **Ash Ketchum** en la parte inferior de la pantalla, sobre un fondo azul oscuro que resalta los elementos.

- **Controles:** Se utilizan las flechas **Izquierda** y **Derecha** del teclado para mover al personaje lateralmente.
- **Objetivo:** Atrapar los Pokémon "buenos" (como **Pikachu**) que caen del cielo para sumar 100 puntos.
- **Obstáculos:** Se deben esquivar los enemigos como **Gengar** (fantasma morado). Si el jugador los toca, sufre una penalización de puntaje (-100 o -200 puntos).
- **Meta:** Acumular la mayor cantidad de puntos antes de que el temporizador de 20 segundos llegue a cero.

6.2. Mecánicas y Estandarización

Al igual que en el juego anterior, todos los *sprites* (personajes y objetos) fueron recopilados de fuentes abiertas en internet para enriquecer la experiencia visual. El movimiento de estos elementos se gestiona mediante un `QTimer` que refresca la posición cada 16ms, logrando una fluidez de 60 FPS.

Un componente crítico de este desarrollo fue la integración de la clase `GameLogger`. Al terminar la partida, el puntaje final se escribe automáticamente en el archivo `eventos_minijuego1.log` en formato JSON, garantizando la coherencia de datos para todo el sistema "Among Us".

```
# Deteccion de colisiones y actualizacion de puntaje
if label.geometry().intersects(self.player.geometry()):
    self.actualizar_puntaje(puntos, tipo)
    label.deleteLater() # Eliminar objeto de pantalla

# Registro final estandarizado (JSON)
def terminar_juego(self):
    self.logger.log_event("End", "Result", score=self.puntaje)
```

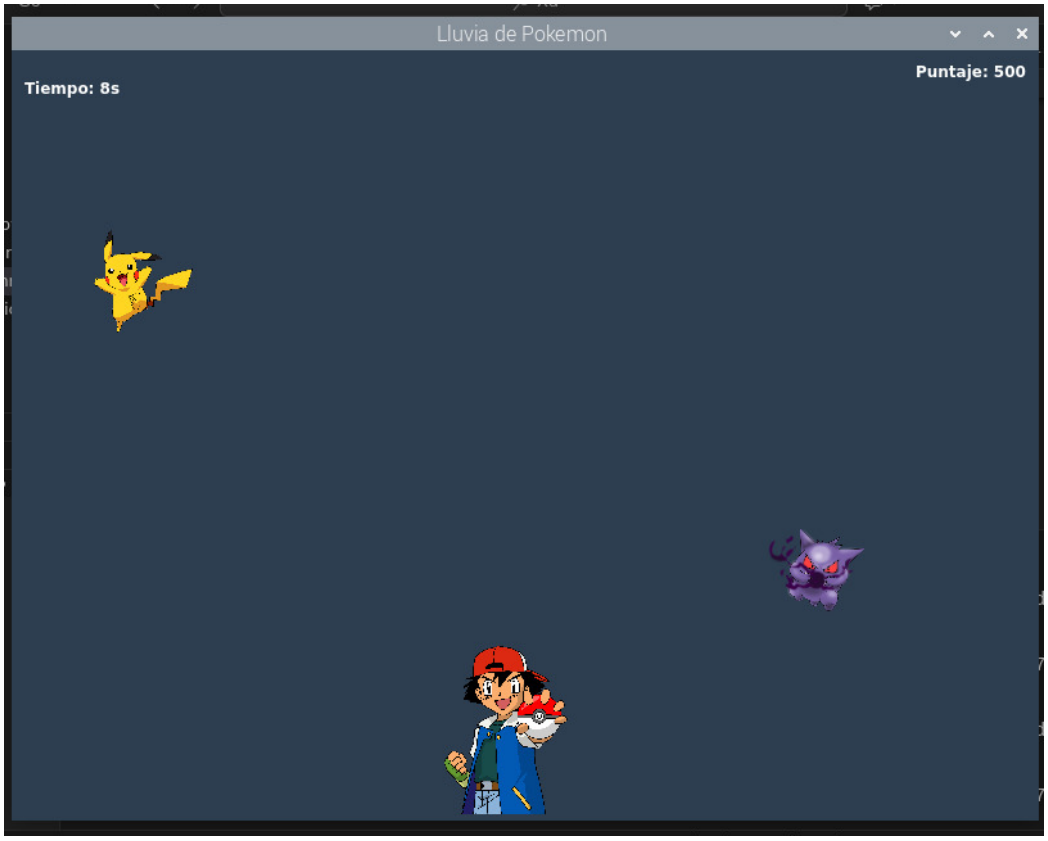

Fig. 5: Juego Lluvia de Pokémon. Se observa a **Ash Ketchum** abajo, un **Pikachu** cayendo a la izquierda y un **Gengar** a la derecha sobre el fondo azul oscuro.

7. Actividad 1.5: Integración y Cliente de Juego

Para unificar los cuatro minijuegos desarrollados y conectarlos con la dinámica central del curso, se implementó un script maestro denominado `main_client.py`. Este programa actúa como el cerebro del "tripulante", gestionando la comunicación con el servidor del profesor (Host Central).

7.1. Arquitectura de Red (SSH/SFTP)

El cliente utiliza la librería **Paramiko** para establecer una conexión segura SSH con la Raspberry Pi del profesor (IP `192.168.0.24`). El flujo de comunicación es bidireccional:

- **Lectura (Polling):** El cliente descarga periódicamente el archivo `game_status.log` del servidor para leer las órdenes (Asignación de tareas o Sabotajes).
- **Escritura (Reporte):** Cada vez que el usuario completa un minijuego, el cliente sube su propio archivo de log local (`player_tonoto.log`) al servidor mediante SFTP, notificando el resultado.

7.2. Lógica de Control y Sabotajes

El script parsea las órdenes recibidas en formato pseudo-JSON. Si recibe la acción `Assign`, identifica el ID del juego y lo ejecuta automáticamente utilizando el comando `os.system`.

Además, se implementó la lógica de **Sabotaje**: si el servidor envía un evento de tipo `Delay`, el cliente congela la ejecución durante una cantidad de segundos específica antes de permitir jugar, simulando una avería en la nave.

```
# Fragmento de main_client.py
if action == "Sabotage" and orden.get("Effect") == "Delay":
    sabotage_pending = orden.get("Value", 0)
    print(f"ALERTA: Sabotaje recibido. Retraso de {sabotage_pending}s.")

if action == "Assign":
    game_id = orden.get("GameID")
    # Ejecutar el script del minijuego correspondiente
    ejecutar_minijuego(game_id, sabotage_delay=sabotage_pending)
```

7.3. Visualización en el Host Central

Gracias a este sistema de reportes estandarizado, el servidor central puede monitorear en tiempo real el estado de todos los jugadores. La Figura 6 muestra la interfaz del profesor, donde se visualizan las barras de vida y el progreso de los tripulantes basándose en los logs que nuestro cliente envía automáticamente.

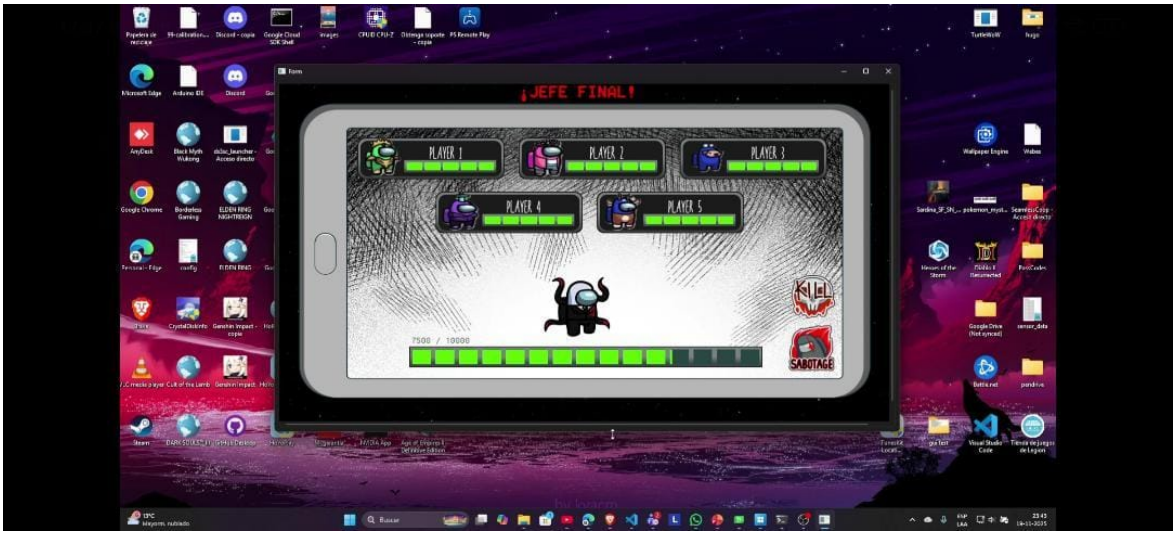

Fig. 6: Vista del Host Central (Profesor). Esta pantalla se actualiza gracias a los archivos de log que nuestro cliente sube vía SFTP tras cada minijuego.

8. Actividad 2: Programando una Pokédex

En esta actividad desarrollamos una aplicación de escritorio utilizando **PyQt6** que simula una Pokédex funcional. El sistema actúa como un puente entre una base de datos digital y el mundo físico, traduciendo los atributos de los Pokémon a señales luminosas mediante la librería `gpiozero`.

8.1. Arquitectura del Software

La interfaz gráfica (GUI) fue diseñada mediante hojas de estilo (QSS) para replicar la estética retro de la consola, utilizando colores `#D32F2F` (Rojo) y fuentes monoespaciadas.

8.1.1. Gestión de Datos y Recursos (Assets Locales)

El programa carga la información desde un archivo `pokedex.json`. Respecto a las imágenes, para garantizar un rendimiento instantáneo y evitar latencia de red durante la demostración, se optó por una estrategia de almacenamiento local. Todas las imágenes fueron descargadas previamente y organizadas en el directorio `/Pokemon todas gen`. El script accede a ellas dinámicamente utilizando rutas relativas:

```
# Ruta dinamica compatible con cualquier ubicacion del proyecto
self.img_folder_path = os.path.join(
    os.path.dirname(os.path.abspath(__file__)),
    "Pokemon todas gen"
)
```

8.1.2. Funcionalidad "Random"

Se implementó un botón específico que invoca la función `show_random`. Esta utiliza `random.randint()` para saltar a un índice aleatorio de la lista, permitiendo pruebas rápidas de la respuesta del hardware ante cambios drásticos de tipos elementales.

8.2. Integración de Hardware

El control físico se realiza instanciando objetos de la clase `RGBLED` de `gpiozero`. La configuración de pines GPIO en el código coincide con el montaje físico en la Raspberry Pi 5:

- **LED Izquierdo (GPIO 26, 6, 5):** Representa el Tipo Primario.
- **LED Derecho (GPIO 17, 27, 22):** Representa el Tipo Secundario.

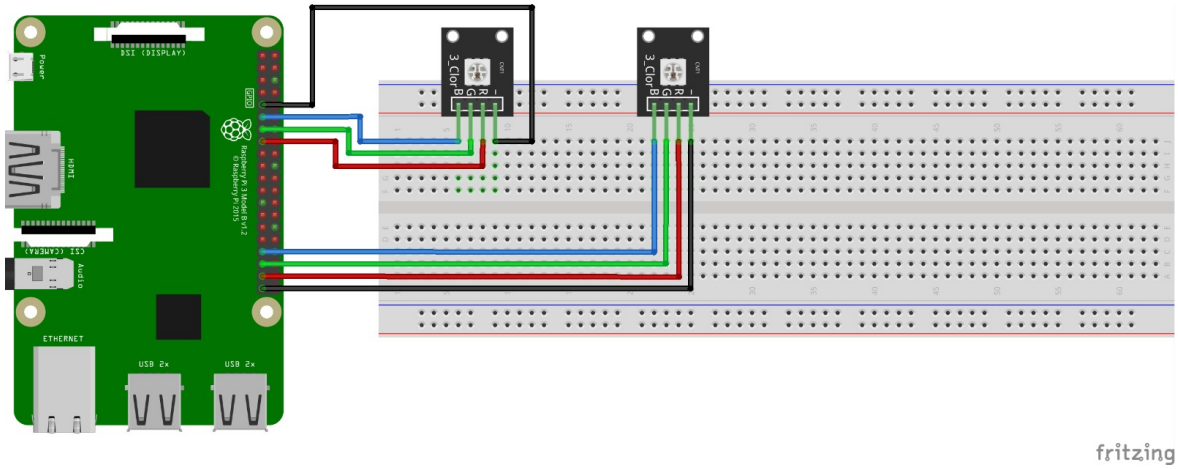

Fig. 7: Esquemático del circuito. Los cátodos comunes de los LEDs RGB se conectan a GND y los ánodos a los pines GPIO definidos en el script.

8.3. Lógica de Mapeo de Colores

El método `update_leds` contiene un diccionario que mapea los tipos (strings) a tuplas RGB normalizadas (0.0 a 1.0). El algoritmo maneja la dualidad de tipos:

```
# Diccionario de colores (Extracto)
colores = {
    "fuego": (1.0, 0.0, 0.0), "planta": (0.0, 1.0, 0.0),
    "agua":  (0.0, 0.0, 1.0), "electrico": (1.0, 1.0, 0.0),
    # ... resto de tipos
}

# Logica de asignacion
if len(tipos_procesados) == 1:
    # Tipo Unico: Ambos LEDs brillan igual
    c1 = c2 = colores.get(tipo, (0.5, 0.5, 0.5))

elif len(tipos_procesados) >= 2:
    # Tipo Doble: Cada LED muestra un tipo
    c1 = colores.get(tipos_procesados[0])
    c2 = colores.get(tipos_procesados[1])

led_izquierda.color = c1
led_derecha.color = c2
```

8.4. Galería de Resultados: Visualización de Tipos Dobles

En estas pruebas se verifica que el sistema separa correctamente los colores cuando el Pokémon posee dos atributos elementales.

Fig. 8: **GUI:** Coalossal (Roca/Fuego).

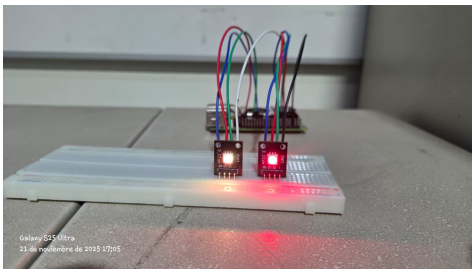

Fig. 9: **Hardware:** LED Izq Ámbar / LED Der Rojo.

Fig. 10: **GUI:** Aurorus (Roca/Hielo).

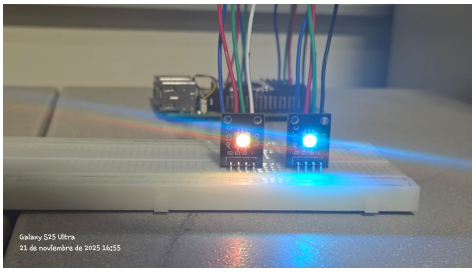

Fig. 11: **Hardware:** LED Izq Ámbar / LED Der Cian.

8.5. Galería de Resultados: Visualización de Tipos Únicos

Cuando el Pokémon es de un solo tipo, el código asigna `c1 = c2`, provocando que ambos LEDs se iluminen del mismo color para reforzar la identidad visual.

Fig. 12: **GUI:** Litten (Tipo Fuego).

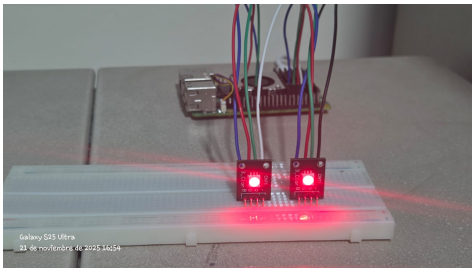

Fig. 13: **Hardware:** Ambos Rojo Intenso.

Fig. 14: **GUI:** Popplio (Tipo Agua).

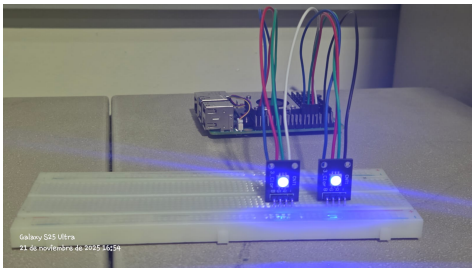

Fig. 15: **Hardware:** Ambos Azul.

Fig. 16: **GUI:** Fomantis (Tipo Planta).

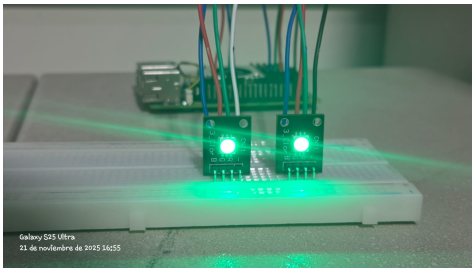

Fig. 17: **Hardware:** Ambos Verde.

9. Conclusión

Este MiniProyecto fue bastante desafiante pero muy entretenido. Logramos convertir la Raspberry Pi en una consola de juegos real, y aprendimos que conectar hardware con software no es tan simple como parece.

Lo más importante que aprendimos fue sobre el manejo de tiempos. Al principio, usábamos procesos de espera simples, pero nos dimos cuenta de que eso congelaba toda la ventana y los botones dejaban de funcionar. Tuvimos que investigar y aprender a usar hilos (`QThread`) para que el juego pudiera "pensar" y "dibujar" al mismo tiempo.

También valoramos mucho la parte eléctrica. El tema del divisor de voltaje para el sensor ultrasónico fue clave; si no lo hubiéramos dimensionado correctamente, habríamos comprometido la integridad del dispositivo. Fue bueno ver cómo la teoría de circuitos se aplica para garantizar la fiabilidad de las señales.

Por último, la creación de módulos de juego separados (físicos y virtuales) y la exitosa implementación de la **Pokédex con doble LED** validaron la eficiencia de nuestro diseño modular. Nos dio la satisfacción de saber que el sistema logró cumplir con la complejidad técnica y estructural solicitada por el enunciado.

Anexo A: Solución de Conectividad desde Casa

Debido a las restricciones de acceso físico al laboratorio durante el desarrollo final del proyecto, nuestro principal desafío fue encontrar una estrategia para conectarla a la red Wi-Fi fuera de la universidad y así habilitar el acceso remoto por VNC.

Estrategia de Conexión

Para lograr esto, decidimos replicar el entorno de red del laboratorio, ya que la Raspberry Pi estaba configurada para conectarse automáticamente a la Wi-Fi de la universidad.

1. Configuramos un *Hotspot* (punto de acceso móvil) en un teléfono, usando el mismo **Nombre de Red** ("LAB.SISTEMAS DE COMUNICACIONES") y la misma contraseña que la sala.
2. Al encender la Raspberry Pi, esta se conectó automáticamente al celular.
3. **Conectamos nuestro computador personal a esa misma red Wi-Fi (el Hotspot) para que la Raspberry Pi y el PC estuvieran en la misma red local.**
4. Como sabíamos que el nombre de nuestra Raspberry es **"squirtle"**, usamos ese *hostname* para conectarnos directamente por VNC desde el PC.

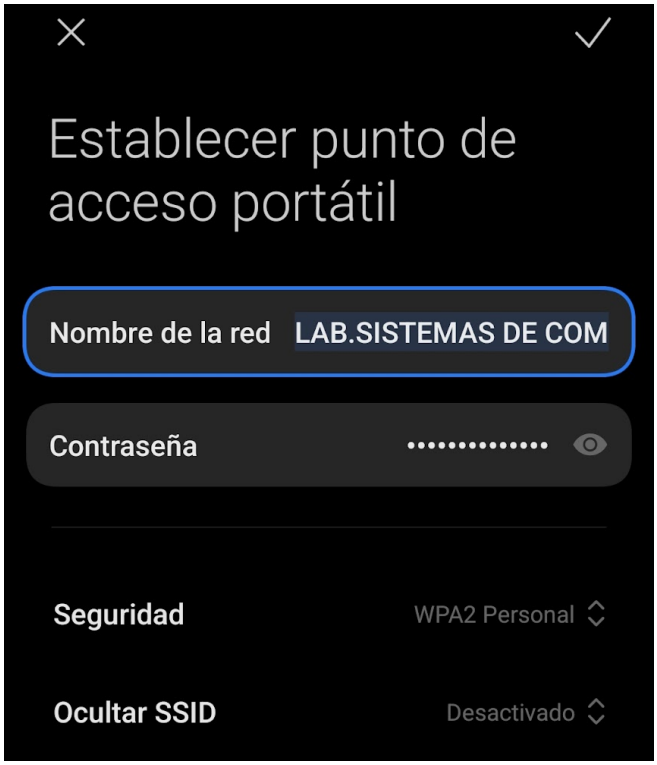

Fig. 18: Configuración del Hotspot móvil replicando las credenciales exactas de la red del laboratorio.

Resultado

Este método funcionó a la perfección. Logramos establecer una conexión **VNC** estable y acceder al escritorio remoto. Además, la conexión de red también habilitó el uso de **FileZilla** (SFTP), lo que

nos permitió transferir los archivos del proyecto desde la Raspberry Pi a nuestro computador de forma eficiente.

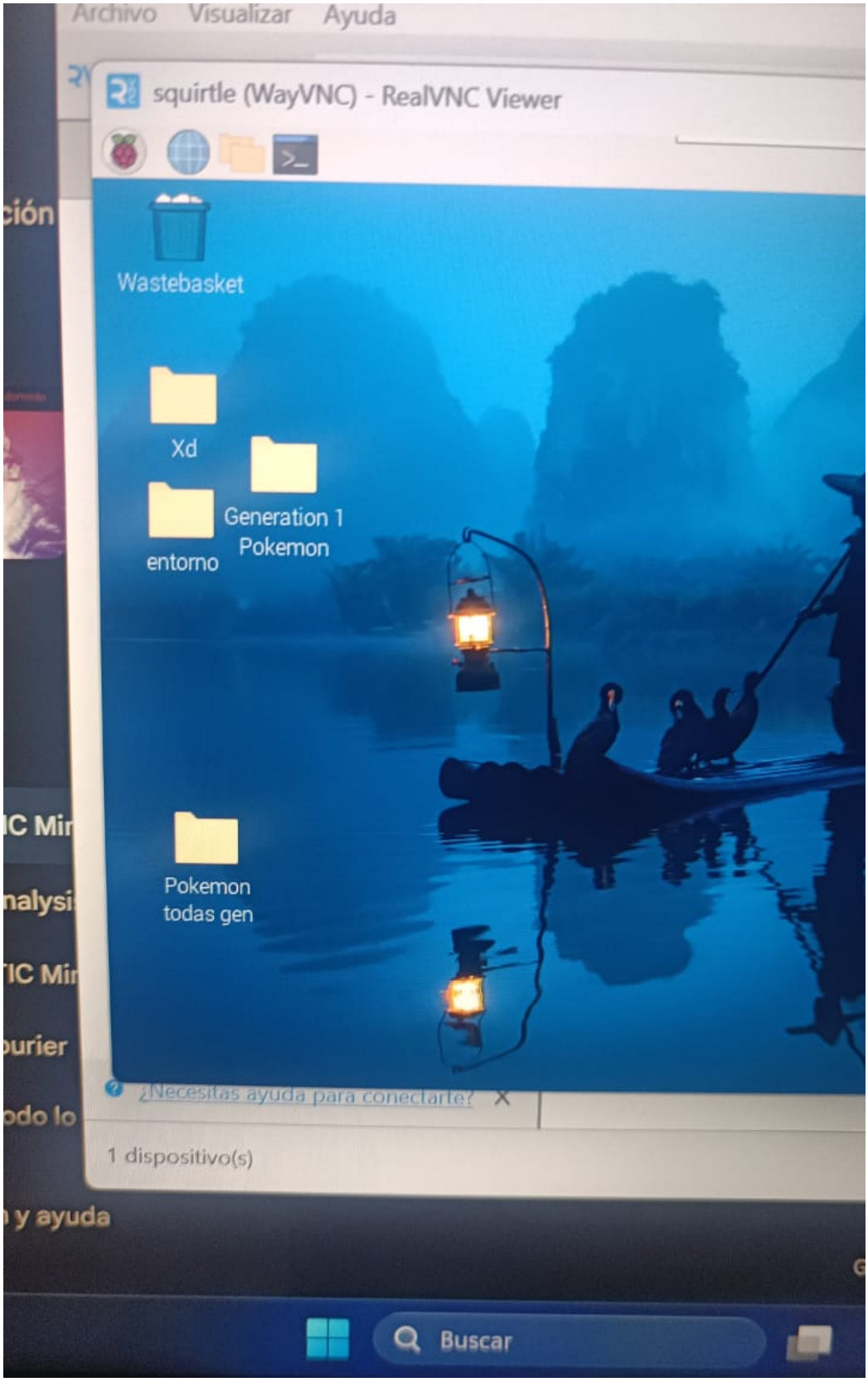

Fig. 19: Escritorio de la Raspberry Pi accedido remotamente