**UNIVERSIDAD DE CONCEPCIÓN**

FACULTAD DE INGENIERÍA

DEPARTAMENTO DE INGENIERÍA ELÉCTRICA



**549253-1 Taller de Aplicación TIC 1**

Vincenzo Caro

Rodrigo Hernández

**Mini Proyecto 2**

Benjamín Barrueto

Joaquín Lozano

Benjamín Pacheco

**Resumen**

El presente informe describe el desarrollo e implementación de un sistema de juegos distribuidos basado en la plataforma Raspberry Pi, en el contexto del Mini Proyecto 2. El trabajo se dividió en dos actividades principales: primero, la creación de "TIC is Among Us", un sistema multijugador cliente-servidor que utiliza protocolos de comunicación asíncrona (archivos de texto planos transferidos vía SSH) para coordinar rondas de minijuegos y eventos de sabotaje entre un Host central y los jugadores. Segundo, el desarrollo de una "Pokédex" interactiva utilizando interfaz gráfica (PyQt5), bases de datos locales (JSON) y control de hardware (LED RGB) para la visualización de datos. El proyecto integra conocimientos de programación orientada a eventos, manejo de sensores GPIO, interfaces gráficas y redes.

.

**Tabla de Contenidos**

[Actividad n°1. TIC is Among Us 1](#_Toc215416755)

[Ítem 1.1 Diseño de minijuegos 1](#_Toc215416756)

[1.1.a) Presentación de Minijuegos del grupo 1](#_Toc215416757)

[1.1.b) Descripción de códigos 2](#_Toc215416758)

[1.1.c) Esquemático del Circuito 5](#_Toc215416759)

[1.1.d) Detalles del Código Implementado 8](#_Toc215416760)

[Actividad n°2. Programando una Pokédex 12](#_Toc215416761)

[2.1.a) Estructura de los Archivos 12](#_Toc215416762)

[2.1.b) Diseño de la Base de Datos (json) 13](#_Toc215416763)

[2.1.c) Detalles del código implementado 13](#_Toc215416764)

[Resultados y Comentarios 14](#_Toc215416765)

# TIC is Among Us

## Diseño de minijuegos

### Presentación de Minijuegos del grupo

Se nos pide diseñar los minijuegos que vamos a utilizar en las rondas 1 y 2, implementando al menos dos minijuegos por consola y dos con utilización de sensores físicos con máxima duración de 15 segundos. Para esto nos basamos en lo realizado en el miniproyecto anterior donde aprendimos conceptos clave sobre como programar sensores, como funciona una proto board, como usar gpiozero, entre muchos otros. Por ende, los siguientes minijuegos se presentan como una mezcolanza de las ideas aprendidas a lo largo del ramo, imaginación e inteligencia artificial.

1. Para el primer juego es de tipeo rápido solo usando la consola, llamado “juego letras”, donde se selecciona al azar una frase dentro de una lista previamente definida, el jugador debe escribirla a lo más en una duración de 15 segundos, el programa captura la entrada en tiempo real midiendo letras correctas, errores y si se acaba el tiempo.
2. El juego siguiente es llamado “juego arco y flecha” el cual es una réplica del estilo de juegos arcade tipo friv.com o juegos.com donde el jugador solo debe apuntar a un objetivo y ser capaz de dar en el blanco una x cantidad de veces para que sea reconocida la victoria. Usamos Pygame para desarrollar una interfaz compuesta por imágenes elegidas por el grupo, en este caso basada en Fondo de Bikini de Bob Esponja.
3. El tercer juego denominado “juego topos”, es basado en juegos de feria que hemos visto en series americanas donde hay que golpear topos de plastico que salen de la máquina antes de que bajen, en este caso utilizamos un led RGB para hacer 4 “topos” uno rojo, otro azul, otro blanco y otro amarillo y botones de esos colores para golpear su “topo” correspondiente.
4. Para el último juego desarrollado por el grupo tenemos el “juego sensor”, claramente utilizamos un sensor ultrasónico conectado a la Raspberry Pi, para el juego la consola pedirá una medición que el grupo (pensando en 3 personas por grupo) deberá tratar de acertar colocando su mano donde piensa que se encuentra la medida que el código arrojó. Finalmente, el jugador con menor error gana la partida.

### Descripción de códigos

* + - Juego de letras:
* Para este código se implementó la función personalizada **input\_con\_tiempo\_real()**, esta función modifica temporalmente los atributos del terminal utilizando las librerías termios y tty, para lograr una interacción en tiempo real.
* El terminal se cambia a una modalidad "no canónica" (raw mode) mediante **tty.setcbreak**(). Esto deshabilita el bufferde línea estándar, permitiendo que el programa capture y procese cada carácter individual **sys.stdin.read(1)** en el instante exacto en que se presiona la tecla, sin necesidad de esperar la confirmación con "Enter".
* Para implementar el límite de tiempo estricto de 15 segundos, se utiliza la función **select.select()**. Esta rutina monitorea el flujo de entrada (stdin) con un temporizador (timeout), si el tiempo expira antes de que el usuario termine de escribir, la función interrumpe el bucle de lectura forzosamente, retornando el texto parcial ingresado hasta ese momento.
* Debido a que el modo rawelimina las funciones de edición nativas de la consola, se implementó manualmente la lógica de la tecla de retroceso (Backspace, código ASCII **\x7f**), manipulando el bufferde texto en memoria y enviando secuencias de escape a la consola para borrar visualmente los caracteres erróneos.
* Finalmente, el código restaura los atributos originales del terminal utilizando **termios.tcsetattr** para evitar errores gráficos en el sistema operativo y procede a calcular el puntaje comparando la cadena del usuario con la frase objetivo, asignando el estado de victoria o derrota según la precisión obtenida.

2)Juego del arco y flecha:

* Este módulo fue desarrollado utilizando la librería **pygame**, lo que permitió crear una interfaz gráfica interactiva con una tasa de refresco de 60 FPS (clock.tick(60)), gestionando la renderización de imágenes (sprites) y la detección de eventos de usuario en tiempo real.
* Para la mecánica de apuntado, se implementó un cálculo trigonométrico vectorial. El ángulo de disparo se determina en cada frame calculando el arco tangente **math.atan2** de la diferencia entre la posición del cursor del mouse y la posición fija del arco.
* La física del proyectil se basa en la descomposición del vector de velocidad. Al disparar, la velocidad total se divide en componentes rectangulares (Vx y Vy) utilizando las funciones **math.cos** y **math.sin** respectivamente, lo que permite que la flecha siga una trayectoria lineal acorde al ángulo seleccionado.
* La detección de impactos se realiza mediante el método **.colliderect()** de la clase **Rect** de Pygame. Este método evalúa en cada ciclo si el rectángulo delimitador (hitbox) de la flecha se intersecta con el rectángulo del objetivo móvil.
* El ciclo de vida del juego está condicionado por dos variables de control: un temporizador global basado en **time.time()** que limita la partida a 15 segundos, y un contador de puntaje que finaliza el juego anticipadamente si se alcanza la condición de victoria (7 aciertos).

3)Juego topos:

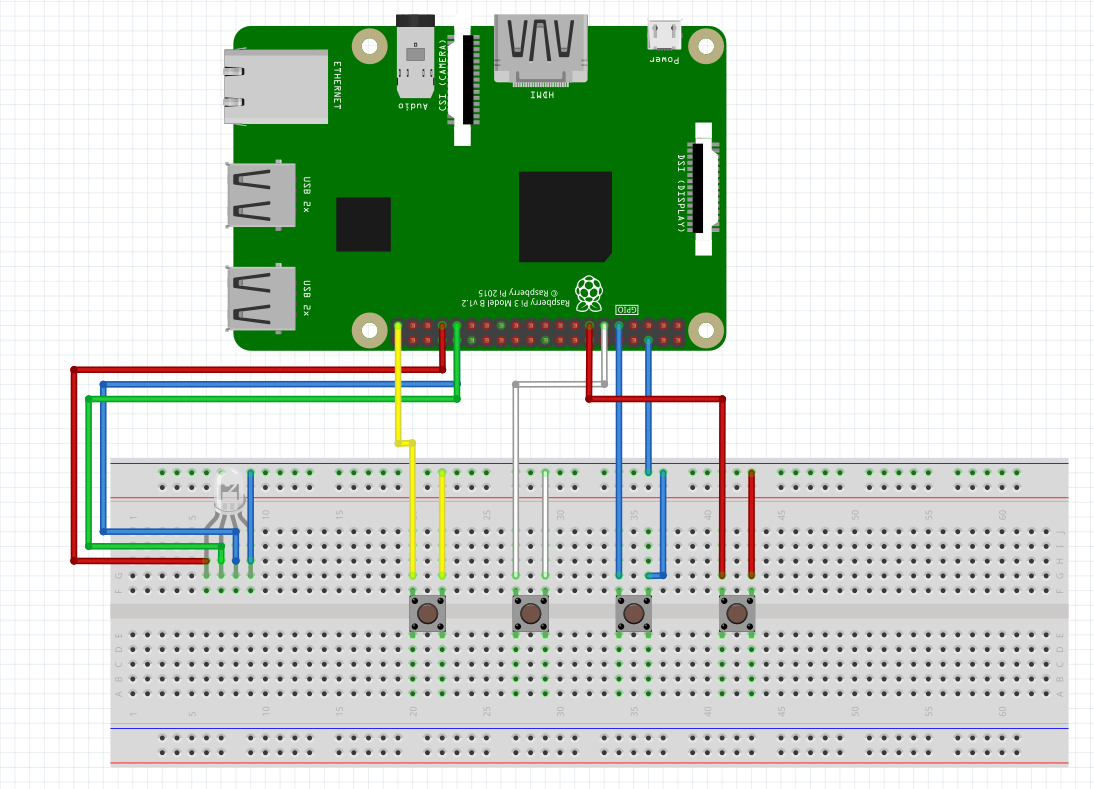
* Utilizamos la librería **gpiozero** para el control de hardware. La lógica se basa en un diccionario de mapeo (COLORES) que vincula tuplas de valores RGB (1,0,0) con objetos de la clase **Button.**
* El flujo del juego se ejecuta dentro de un bucle de tiempo controlado **while time.time()-inicio\_inicio\_juego < 15.0**, garantizando que la actividad se ajuste estrictamente a la ventana de tiempo asignada.
* En cada iteración, el algoritmo selecciona aleatoriamente un par {Color, Botón} utilizando **random.choice.** Acto seguido, el LED RGB se ilumina con el color seleccionado y se inicia un temporizador de reacción (DURACION\_COLOR=0.5).
* La detección de entrada se realiza mediante un sondeo continuo (polling) del estado de los botones (**boton.is\_pressed)**. Si el usuario presiona el botón correcto dentro de la ventana de tiempo, se incrementa el contador de aciertos y el puntaje (+12.5 puntos), de lo contrario, o si se presiona un botón erróneo, el sistema registra el fallo sin sumar puntos.
* Al finalizar el tiempo se registran los datos, si el usuario logra al menos 5 aciertos, se declara “VICTORIA”, en caso contrario, dirá “DERROTA”.

4)Juego del sensor:

* Este juego implementa un sistema de medición física utilizando el sensor ultrasónico KY-050 ULTRASONIC DISTANCE SENSOR MODULE, gestionado por la clase **DistanceSensor** de la librería gpiozero.
* Para mitigar el ruido eléctrico y las lecturas erráticas típicas de los sensores de ultrasonido, se desarrolló la función de filtrado **read\_once()**. Esta rutina no toma una sola medición, sino que captura una ráfaga de muestras consecutivas (por defecto 7 muestras con un intervalo de 0.04s), descarta los valores fuera de rango (filtros de 3 cm a 350 cm) y calcula la mediana estadística **statistics.median** para obtener un valor final estable y confiable.
* La mecánica del juego consiste en un desafío de precisión propioceptiva. El sistema genera un **objetivo aleatorio** (random.uniform) entre 15 cm y 100 cm, y concede al jugador una ventana de tiempo (5 segundos) para posicionar su mano a la distancia estimada sin ayudas visuales inmediatas.
* Al finalizar la cuenta regresiva, el sistema captura la distancia real y calcula el **error ebsoluto** (|Medicion - Objetivo|).
* Finalmente, el algoritmo evalúa el desempeño mediante una lógica competitiva post ronda. Una vez completados todos los intentos, el sistema analiza la lista de resultados y determina quién obtuvo la menor distancia de error **(min())** respecto al objetivo. El estado de Victoria se le asigna exclusivamente a dicho participante, mientras que el resto recibe una Derrota.

### Esquemático del Circuito

A continuación se presentan los esquemas de los circuitos que implementamos en nuestros minjuegos en los que se hizo uso del Kit de Leds, en este caso para el juego de “Topos” y para el juego de medir distancia o el juego del “sensor”.



*Figura 1. Fritzing del Juego de Topos*

Conexiones GPIO para nuestro juego del “Topo:

LED RGB

|  |  |
| --- | --- |
| KY-009 | Raspberry Pi |
| R | GPIO13 |
| G | GPIO6 |
| B | GPIO12 |
| GND | GND |

Botón 1 (Amarillo)

|  |  |
| --- | --- |
| Arcade Pushbutton | Raspberry Pi |
| LEFT | GPIO21 |
| RIGHT | GND |

Botón 2 (Blanco)

|  |  |
| --- | --- |
| Arcade Pushbutton | Raspberry Pi |
| LEFT | GPIO17 |
| RIGHT | GND |

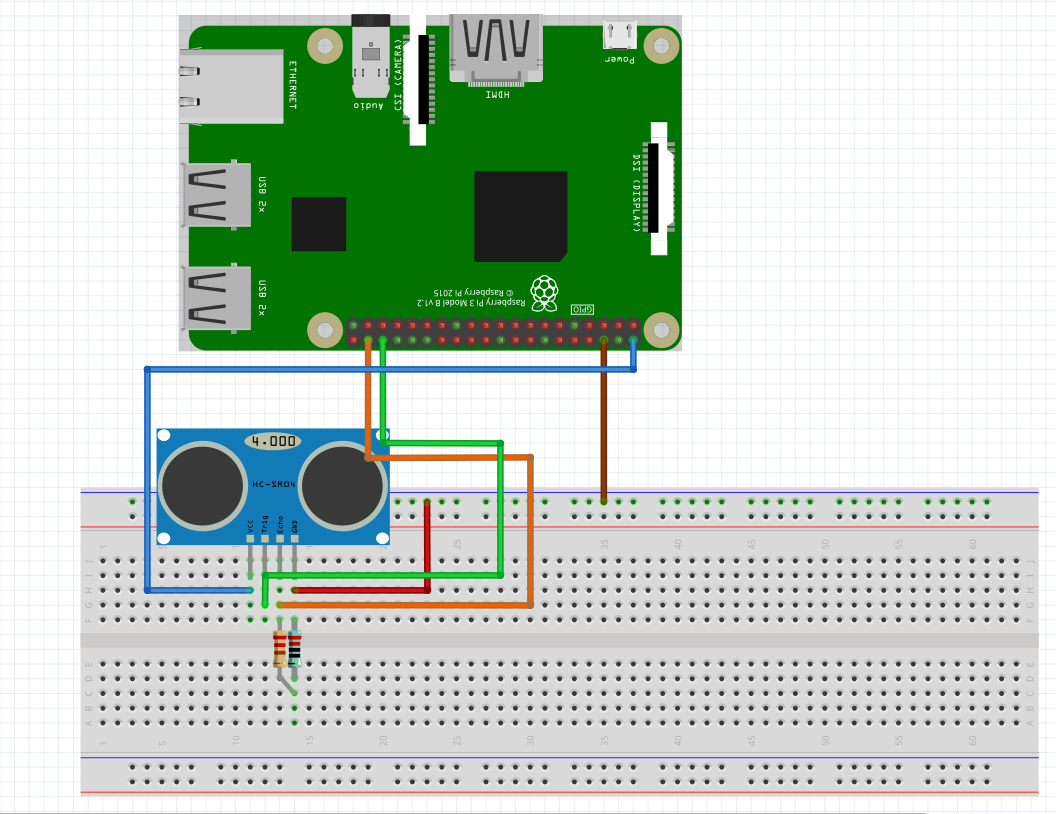
Botón 3 (Azul)

|  |  |
| --- | --- |
| Arcade Pushbutton | Raspberry Pi |
| LEFT | GPIO18 |
| RIGHT | GND |

Botón 4 (Rojo)

|  |  |
| --- | --- |
| Arcade Pushbutton | Raspberry Pi |
| LEFT | GPIO27 |
| RIGHT | GND |

Para el juego del sensor:



*Figura 2. Fritzing del Juego del Sensor*

Conexiones GPIO juego para medir distancia o del “sensor”:

Sensor de Distancia

|  |  |
| --- | --- |
| KY-050 | Raspberry Pi |
| VCC | 5V |
| Trig | GPIO16 |
| Echo | GPIO20 |
| GND | GND |

### Detalles del Código Implementado

Para cada juego, los separamos en sus partes más importantes, lo presentado acá sigue la estructura lógica de cada juego:

**Juego 1: Escribe lo más rápido que puedas**

1. Inicio:  
    Se cargan las constantes principales, como las rutas de los archivos de registro, la lista de frases y el tiempo máximo permitido.
2. Registro con host (Lobby):  
    Se escribe en player\_events.log la acción **Join**, luego el programa lee desde game\_status.log la confirmación **Accepted** enviada por el host, y finalmente se registra **Ready**.  
    Con esto el jugador queda habilitado para comenzar el juego.
3. Lectura de instrucciones del host  
   El programa lee la última asignación (Assign) para conocer el stage y el GameID, y después revisa si existe un sabotaje activo, obteniendo su tipo (Effect) y valor (Value).
4. Aplicación de sabotajes antes de jugar:
   1. **Disable:** Se registra TimeOut y el programa finaliza.
   2. **Delay:** Se reduce el tiempo máximo de tipeo (15 s menos el valor del sabotaje, con un mínimo de 5 s).
   3. **ScoreSteal:** Se aplica después del cálculo de puntaje, al final.
5. Minijuego:  
    Se selecciona aleatoriamente una frase que el jugador debe escribir exactamente igual.  
    Luego, el jugador presiona ENTER para comenzar y dispone del tiempo establecido.  
    Se calcula el **score** como precisión menos penalización por errores, limitado al rango 0–100.  
    El resultado final puede ser **Win**, **Lose** o **TimeOut**, dependiendo de la precisión y el tiempo.
6. Post–juego y registro:  
    Si había sabotaje del tipo ScoreSteal, se descuenta el porcentaje indicado del puntaje anterior.  
    Finalmente se guarda en el archivo player\_events.log una línea con el formato:
7. timestamp stage PlayerID Action:"Ready" GameID Result Score
8. y el programa termina.

**Juego 2: Arco y flecha**

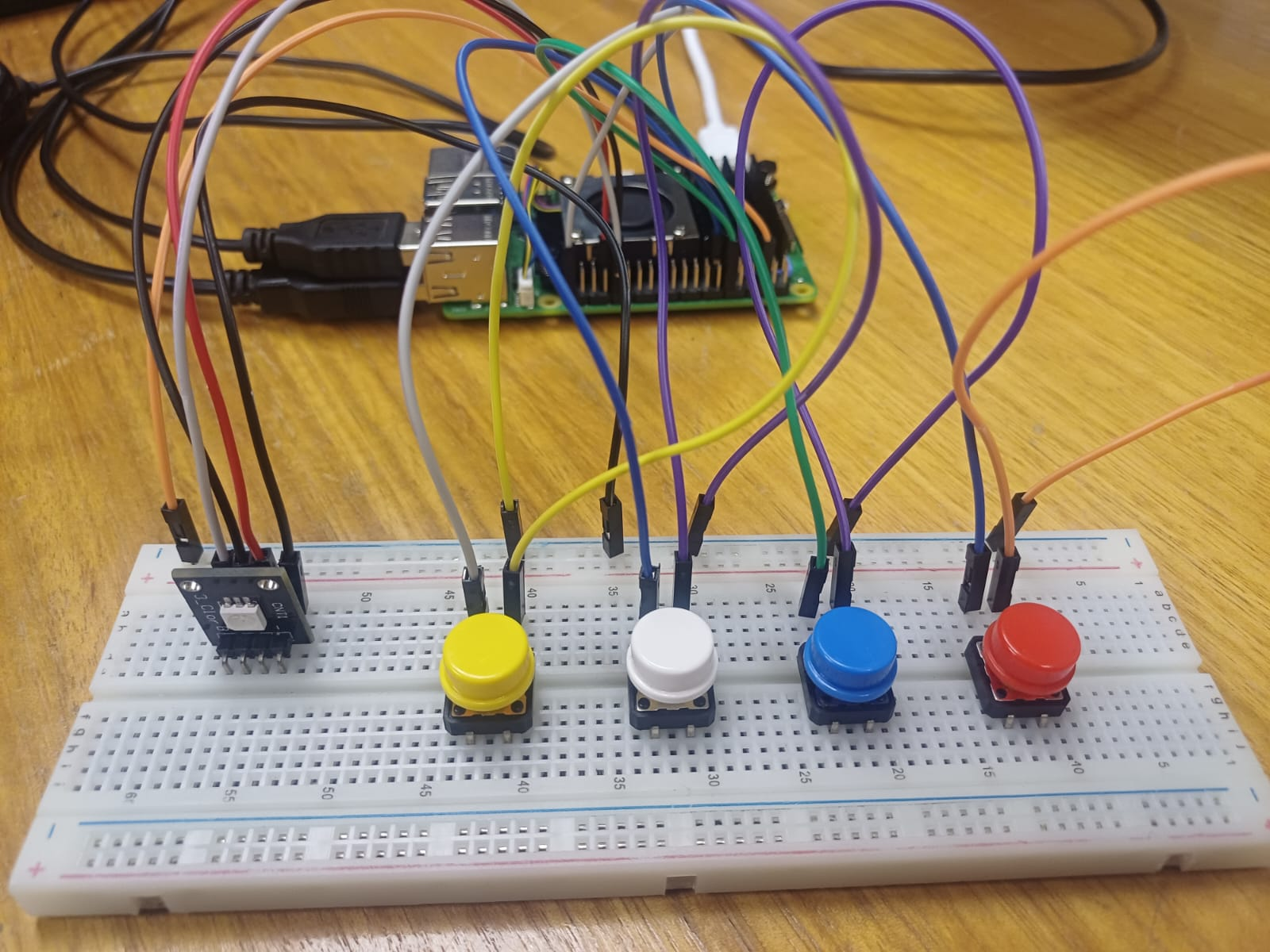
1. Inicio:  
    Se inicializa la librería Pygame, se crean las variables del juego y se cargan las imágenes (arco, flecha y el objetivo).
2. Registro con el host:  
    Se realiza la secuencia **Join → Accepted → Ready**, dejando al jugador en el lobby.
3. Lectura de instrucciones del host:  
    Se lee el Assign para conocer GAME\_STAGE y GAME\_ID, y luego se verifica si hay sabotaje activo.
4. Aplicación de sabotajes:
   1. **Disable:** Se registra TimeOut y el juego no se ejecuta.
   2. **Delay:** Se reduce el tiempo total disponible (15 s menos el valor del sabotaje).
   3. **ScoreSteal:** Se aplicará después del cálculo del puntaje final.
5. Minijuego:  
    El jugador controla el arco y dispara flechas presionando la tecla ESPACIO.  
    Se actualizan las posiciones de los objetos y se detectan colisiones entre la flecha y el blanco.  
    Cada acierto suma puntos, y el juego termina cuando se cumple el tiempo límite o se alcanza la cantidad de aciertos requeridos.
6. Resultado y registro:  
    Dependiendo del puntaje o del tiempo restante, se define el resultado como **“Victoria”** o **“Tiempo agotado”**.  
    Si existe ScoreSteal, se descuenta el porcentaje indicado.  
    Finalmente se guarda el registro en player\_events.log.

**Juego 3: Topos**

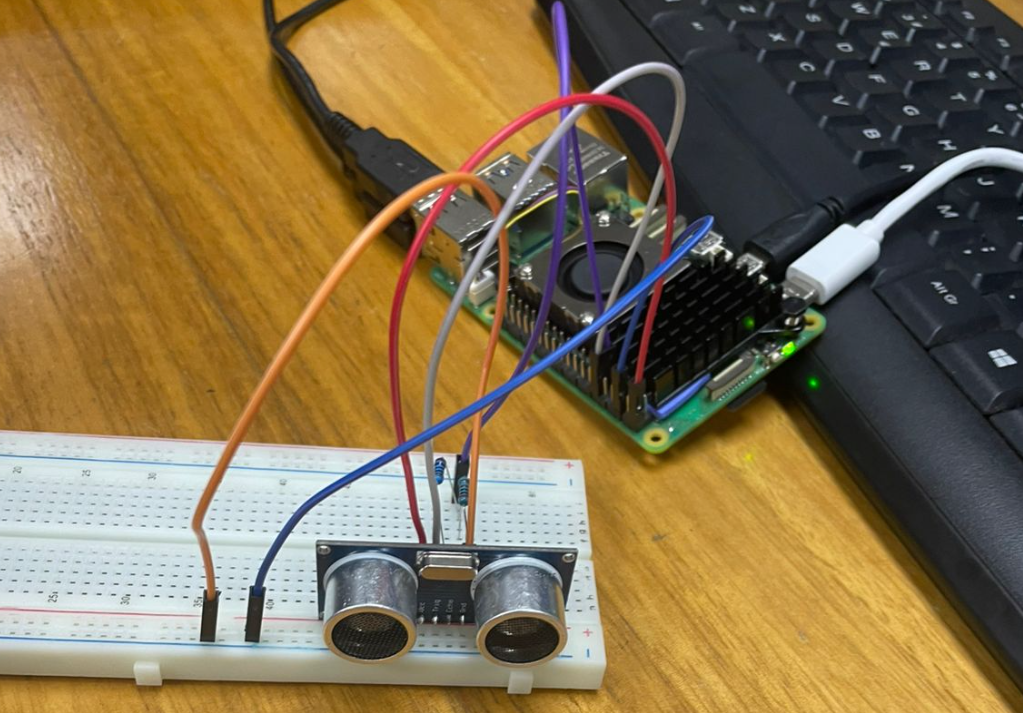
1. Inicio:  
    Se configuran los pines de la Raspberry Pi, el LED RGB y los botones.  
    Se define la correspondencia entre cada color y su botón correcto.
2. Registro con el host:  
    Se envían las acciones **Join**, **Accepted** y **Ready**.
3. Lectura de instrucciones del host  
    Se lee el Assign para conocer GAME\_STAGE y GAME\_ID, y luego se revisa si hay sabotaje activo.
4. Aplicación de sabotajes:
   1. **Disable:** Se registra TimeOut y no se ejecuta el juego.
   2. **Delay:** Se reduce el tiempo total del juego (15 s menos el valor del sabotaje, con un mínimo de 5 s).
   3. **ScoreSteal:** Se aplicará al final sobre el puntaje obtenido.
5. Minijuego:  
    Se ejecuta una secuencia inicial de prueba donde se encienden los cuatro colores para verificar el LED.  
    Durante el tiempo disponible, el LED muestra colores aleatorios y el jugador debe presionar el botón correspondiente en menos de medio segundo.  
    Si acierta, suma puntos; si se equivoca, solo se muestra un mensaje de error.  
    Al finalizar, si el número de aciertos es suficiente, se muestra “Victoria”; si no, “Derrota”.
6. Pos–juego y registro:  
    Si hay sabotaje ScoreSteal, se aplica la penalización.  
    Luego se guarda el resultado en player\_events.log y se liberan los pines del hardware.

**Juego 4: Sensor de distancia**

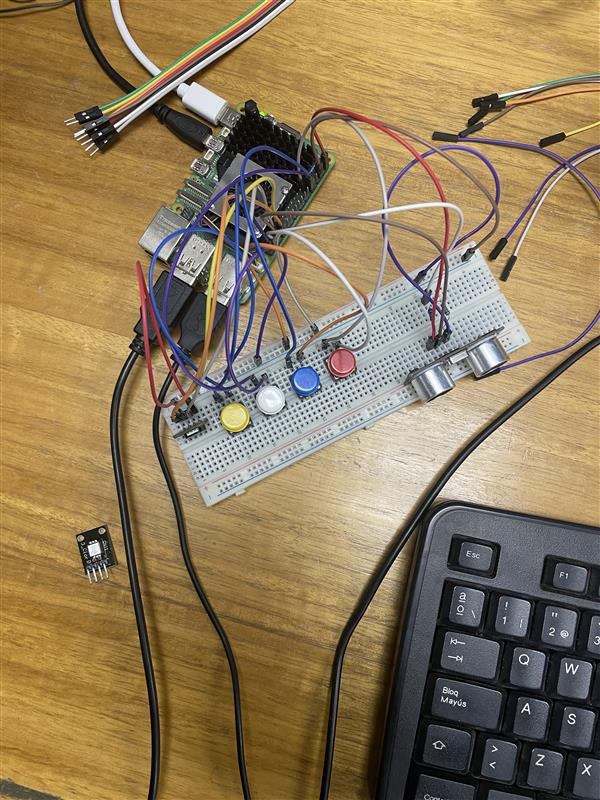
1. Inicio:  
    Se inicializa el sensor ultrasónico y se definen los rangos válidos de distancia (por ejemplo, entre 15 y 50 cm).
2. Registro con el host:  
    Se envían las acciones **Join**, **Accepted** y **Ready**.
3. Lectura de instrucciones del host:  
    Se obtiene el Assign con GAME\_STAGE y GAME\_ID, y se verifica si hay sabotaje activo.
4. Aplicación de sabotajes:
   1. **Disable:** Se registra TimeOut y se finaliza.
   2. **Delay:** Se reduce el tiempo máximo permitido para realizar la medición.
   3. **ScoreSteal:** Se aplicará sobre el puntaje final.
5. Minijuego:  
    Se genera una distancia objetivo-aleatoria y se muestra al jugador.  
    El jugador tiene una sola oportunidad para colocar la mano frente al sensor intentando coincidir con la distancia indicada.  
    Se toma una medición y se calcula la diferencia entre la lectura y el objetivo.  
    Con base en esa diferencia se calcula el puntaje y el resultado: **Win**, **Lose** o **TimeOut**.
6. Post–juego y registro:  
    Se aplica ScoreSteal si corresponde y se guarda el resultado en player\_events.log con el formato habitual.  
    El programa finaliza.



*Figura 3. Fotografía del protoboard del Juego de Topos*



*Figura 4. Fotografía del protoboard del Juego del Sensor*



*Figura 5. Fotografía del protoboard con los sensores de los 2 juegos al momento de conectarse a la Raspberry del profesor en clases. (Se adjunta el código en los anexos)*

# Programando una Pokédex

El objetivo de esta segunda actividad es el desarrollo de una Pokédex Digital Interactiva. El sistema combina una interfaz gráfica de usuario (GUI) moderna desarrollada en Python con interacción de hardware físico, permitiendo visualizar información detallada de diversos Pokémon y obteniendo retroalimentación visual a través de iluminación LED RGB sincronizada con el tipo de Pokémon mostrado.

### Estructura de los Archivos

Para tener un buen orden en esta actividad, creamos una carpeta para separar la lógica de programación de los recursos gráficos y base de datos.

* Modulo principal (pokedex.py): Script ejecutable que contiene la lógica de la aplicación, la gestión de la interfaz y el control de hardware.
* Base de datos (pokemons.json): Archivo de texto estructurado que almacena la información de cada Pokémon.
* Carpeta de PNGs: Carpeta dedicada al almacenamiento de las imágenes en formato PNG de cada pokemon.

### Diseño de la Base de Datos (json)

La gestión de datos se implementó utilizando el formato JSON (JavaScript Object Notation). Esta elección se justifica por su ligereza y su integración nativa con los diccionarios de Python.  
El esquema de datos para cada entrada (Pokémon) consta de los siguientes atributos:

* ID: Identificador único (ej. "001"), mantenido como cadena de texto para preservar el formato visual clásico.
* Metadatos: Campos para nombre, tipo y descripción que alimentan la interfaz visual.
* Referencia de Imagen: Ruta relativa (ej. "pokemon/Bulbasur.png") que permite al script cargar dinámicamente el recurso gráfico sin depender de rutas absolutas fijas.

### Detalles del código implementado

El desarrollo de la interfaz gráfica y la lógica de control se llevó a cabo utilizando Python y la librería PyQt5, estructurando la aplicación para cumplir con los requisitos funcionales de visualización y navegación.

1. Configuración de Hardware: Al inicio del código, se realiza la importación de las librerías necesarias, destacando PyQt5 para la interfaz y gpiozero para el control físico. Se implementa un bloque de control de excepciones para detectar si el programa se ejecuta en una Raspberry Pi, en caso afirmativo, se inicializa el objeto RGBLED asignando los pines GPIO correspondientes para el control de luces.

2. Inicialización de la Clase y Carga de Datos: La clase principal Pokédex hereda de QWidget. En su constructor (\_\_init\_\_), lo primero que se ejecuta es el método de carga de datos, el cual lee el archivo JSON y almacena la información de los Pokémon en una lista interna, asegurando que los datos estén disponibles antes de construir la interfaz.

3. Construcción de la Interfaz Gráfica (inicializar\_ui): En este método se instancian y organizan los widgets en el orden visual de la pantalla:

* Etiquetas de Información (QLabel): Se definieron múltiples objetos QLabel. Primero, se crearon las etiquetas para el título y el número. Posteriormente, se instanció un QLabel central reservado para contener la imagen del Pokémon, y finalmente, etiquetas para el "Nombre", "Tipo" y "Descripción".
* Controles de Navegación (QPushButton): Al final del layout se añadieron tres botones de la clase QPushButton ("Anterior", "Aleatorio", "Siguiente"). Cada botón se conectó mediante señales a sus funciones respectivas para permitir la navegación por la lista.

4. Lógica de Actualización (actualizar\_pantalla): Este es el método recurrente que se ejecuta cada vez que el usuario interactúa con un botón. Sigue el siguiente orden de operaciones para refrescar la vista:

* Actualización de Texto: Se extraen los datos del Pokémon actual y, utilizando el método setText() de los widgets QLabel, se reemplaza el contenido del número, nombre, tipo y descripción.
* Actualización de Imagen: Se construye la ruta al archivo PNG y se carga en el QLabel de imagen mediante el método setPixmap(), el cual renderiza el gráfico en el área designada.
* Control de Hardware (LED): Finalmente, el código evalúa el "tipo" del Pokémon mostrado (ej. Agua, Fuego). Usando un diccionario de mapeo de colores, envía la señal RGB correspondiente al LED físico, sincronizando así el hardware con la interfaz gráfica.

Resultados y Comentarios

**Respecto de los resultados**

En cuanto a los resultados obtenidos, lamentablemente no fue establecida la conexión de la forma más satisfactoria posible con el host, puesto que el feedback JSON recibido no era el esperado, no obstante, el anfitrión logró detectar el código local y fue posible acceder al lobby del juego. En cuanto al desarrollo de los juegos, fueron descubiertas nuevas bibliotecas y herramientas de Python que permitirán un mejor desplante en futuros proyectos, la utilización del sensor fue un desafío completado exitosamente y el agregado de interfases por medio de PyGame permitió abordar nuevos estilos de juegos o programas que de igual manera facilitan e incentivan la creatividad en este tipo de evaluaciones. Por último, respecto a la actividad 2 se cree que se desarrolló una Pókedex que cumple las expectativas, los colores del Led RGB y los botones de interfaz funcionaron de acuerdo a lo esperado.

**Respecto de los comentarios**

El desarrollo de la actividad presentó un nivel de dificultad acorde a lo esperado, logrando implementar la lógica de los minijuegos y la interfaz gráfica sin mayores contratiempos. Sin embargo, el único desafío significativo que enfrentamos como grupo fue la integración de red con el Host Central (Raspberry Pi del profesor).  
Aunque logramos establecer la conexión SSH y unirnos al lobby del Among Us, nuestro código no lograba leer o procesar correctamente los datos provenientes del archivo game\_status.log del Host. Esto generó una situación en la que, a pesar de estar "conectados", el cliente no reaccionaba a las órdenes de inicio de juego.

##### Información Adicional

En este apartado, adjuntamos el código con el cual hicimos la conexión mediante SSH para la activdad en clases.





