Benemérita Universidad Autónoma de Puebla

Facultad de Ciencias de la Computación

Proyecto:

El Desafío del Gato Extremo

Integrantes:

José Gabriel Rojas Pérez | 202156918

José Ángel Lozano Cruz | 202242437

Giovani Jimenez Bonilla | 202129781

Materia: Arquitectura de Software

Docente: Alfredo García Suárez

Periodo: Otoño 2025

1. Introducción

El siguiente proyecto consiste en desarrollar un juego que permita la interacción humano computadora a través de un juego clásico conocido como tres en raya, tic-tac-toc o juego del gato. Por lo que a través de este juego tiene como intención fomentar el aprendizaje interactivo y divertido en los estudiantes de nivel de primaria a través de la gamificación. Por lo que se busca mejorar la interacción que tiene el usuario con un sistema al utilizar un mando físico diseñado exclusivamente para este juego.

Además, este sistema tendrá una arquitectura de cliente-servidor simple donde a través de un backend hacemos peticiones que brindan en tiempo real las preguntas destinadas para que el usuario responda al momento de jugar, estas se obtienen mediante protocolo HTTP que esta alojada de manera local con la intención de facilitar la carga de preguntas en el juego, estas preguntas son mixtas en base a los conocimientos de nivel primaria con una interfaz atractiva y interactiva que hace que el entendimiento del juego sea más fácil.

En este proyecto se busca que el jugador no solo haga clic con el ratón, sino que se pueda manejar el juego a través de un joystick físico conectado a un ESP32/Arduino. Para lograrlo, el primer paso es contar con un programa sencillo, pero bien estructurado que lea continuamente la posición de la palanca y el estado de los botones, y transforme todo eso en datos que la computadora pueda entender. Justo eso es lo que hace el código presentado: convierte los movimientos y pulsaciones del usuario en una serie de números y etiquetas que viajan por el puerto serie a gran velocidad.

2. Objetivo

Diseñar e implementar una plataforma de interacción humano computadora (HCI) que utilice un juego del gato para mejorar la experiencia de interacción del usuario y fomentar el aprendizaje de manera dinámica, interactiva y divertida.

2.1. Alcance

El sistema permite que dos jugadores compitan en un tablero físico de 3×3 casillas, controlado mediante un microcontrolador ESP32. La captura de entradas se realiza a través de dispositivos físicos (joystick y botones), mientras que la lógica del juego se ejecuta en una aplicación desarrollada en Python, encargada de procesar las jugadas, determinar los resultados y gestionar la comunicación con el ESP32.

3. Dinámica Educativa del Juego

Tres en raya es un juego clásico de mesa para dos jugadores en un tablero de 3x3 casillas. Cada jugador alterna turnos para colocar su símbolo (X o O) en una casilla vacía. El objetivo es formar una línea de tres símbolos idénticos (horizontal, vertical o diagonal). El primero en lograrlo gana; si todas las casillas se llenan sin ganador, es empate.

La mecánica central se adapta para un contexto educativo:

- Antes de realizar un movimiento, el jugador debe responder una pregunta aleatoria de un banco de preguntas.
- Si la respuesta es correcta, el jugador puede colocar su ficha (X o O).
- Si la respuesta es incorrecta, el jugador pierde el turno.
- El banco de preguntas debe contener al menos 20 preguntas.

Las reglas se definen de la siguiente manera:

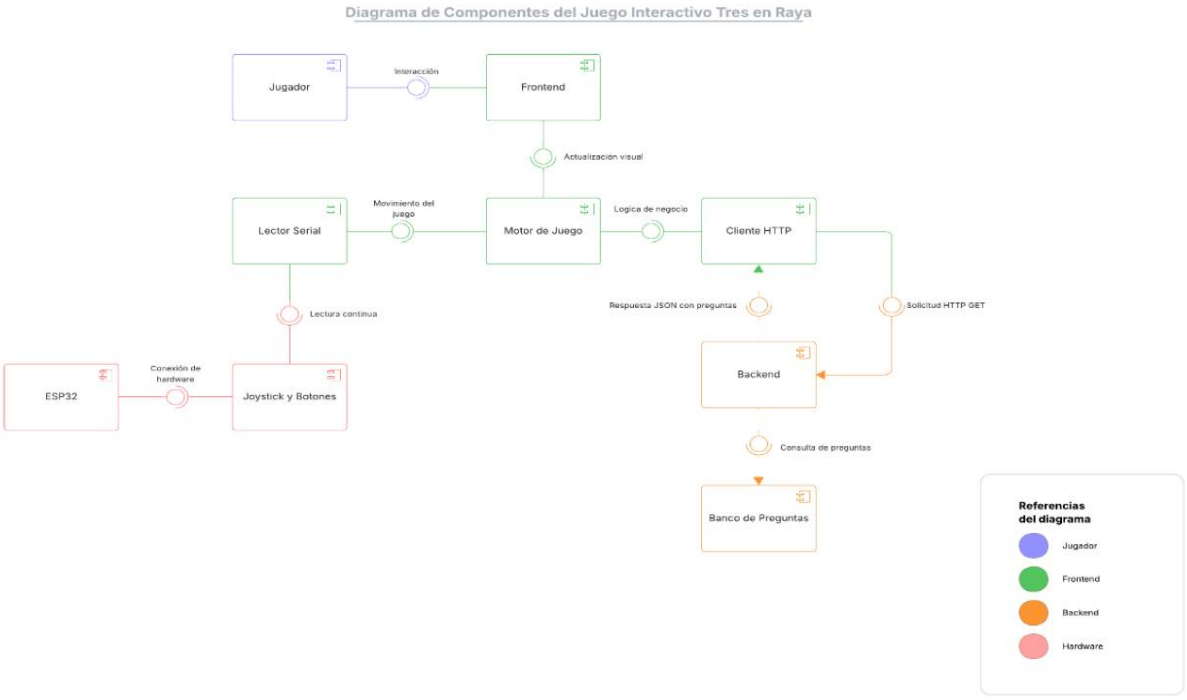
- Los jugadores alternan turnos (Jugador 1: X, Jugador 2: O).
- En cada turno: Selecciona una casilla vacía usando el mando/ESP32.
- Responde una pregunta educativa en la GUI (múltiples opciones o texto).
- Si correcta: Coloca ficha. Si incorrecta: Pierde turno.
- Gana quien forme tres en línea. Empate si tablero lleno.
- Opciones: Modo vs. IA (para un jugador), temas educativos personalizables.

4. Arquitectura del Sistema

4.1. Estilo Arquitectónico: Cliente-Servidor

El sistema adopta un estilo arquitectónico cliente–servidor, en el que la lógica de negocio y la gestión del estado del juego se centralizan en el servidor, mientras que los clientes la interfaz gráfica y el mando físico se encargan de la interacción directa con el usuario. Los clientes envían sus solicitudes y reciben las actualizaciones correspondientes de forma asíncrona, lo que permite una comunicación fluida en tiempo casi real. Esta separación de responsabilidades facilita el mantenimiento y la escalabilidad: el servidor concentra la lógica del juego y los componentes educativos, incluyendo validación de jugadas, control de turnos y evaluación de respuestas, mientras que el cliente se enfoca en la presentación visual, la retroalimentación al jugador y la captura de entradas desde el joystick y los botones. De este modo, cualquier cambio en las reglas del juego o en los contenidos educativos puede realizarse principalmente en el servidor, sin modificar la capa de interacción física o la interfaz gráfica.

4.2. Diagrama de Componentes



La arquitectura Cliente-Servidor se implementa con Python/Flask como el núcleo del servidor y HTML/JavaScript como la capa de presentación:

- Backend:
  - **Tecnología:** Flask (Python). Esto actúa como el Servicio de Preguntas y el punto central de entrada para la GUI.
  - **Servicio Serial Asíncrono:** La comunicación con el hardware ESP32 se maneja en un hilo separado (serial_worker usando threading y la biblioteca serial) para evitar bloquear el servidor web. Esto permite al servidor leer continuamente el estado del Joystick y exponerlo.
  - **Endpoints Expuestos:**
    - @app.route('/get_input'): Provee el estado actual del Joystick (x, y, sw, btn, connected) al Frontend en formato JSON.
    - @app.route('/questions'): Implementa el Generador de Preguntas. Carga el archivo preguntas.json, mezcla las preguntas (random.shuffle), y devuelve un número limitado de ítems en JSON.
- **Frontend:**
  - **Tecnología:** HTML, CSS y JavaScript. Se utiliza un cliente web local que se conecta al servidor Flask (en lugar de una aplicación de escritorio pura Tkinter, como se preveía).

- **Componentes de la UI:** La interfaz incluye el tablero (#board), el cursor de selección (#cursor), y elementos modales (#modal) para mostrar las preguntas y el tiempo límite.
- **Manejo de la Interacción:** El archivo JavaScript (game.js, referenciado en index.html) consume los endpoints del servidor Flask (como /get_input y /questions) para:
- **Actualizar la Posición del Cursor:** Leer el estado del Joystick en tiempo real.
- **Activar la Pregunta:** Mostrar el modal (#modal) con las opciones de la pregunta cargadas del endpoint /questions.

5. Especificaciones Técnicas

5.1. Implementación de Banco de Preguntas

- **Estructura del JSON:** El archivo contiene un arreglo de ítems. Cada ítem es una pregunta (q), con un arreglo de opciones (opts), la respuesta correcta (ans como índice) y una explicación (exp).

- **Contenido:** El banco cumple ampliamente con el requisito de tener "al menos 10 preguntas por tema", conteniendo un total de 36 preguntas de temas mixtos básicos (matemáticas, geografía, historia, lenguaje).

- **Regla Educativa Implementada:** El archivo también incluye el parámetro tiempo_limite_seg (15 segundos), que soporta el requisito funcional de asignar puntuación extra por respuestas correctas rápidas (RF05), al establecer una ventana de tiempo para la respuesta.

5.2. Integración HCI y Estado

La integración del hardware (ESP32) con el software se implementa con las siguientes estructuras de datos y lógica:

- **Modelo de Datos del Mando:** El estado del Joystick se mantiene en el diccionario Python joystick_state, que expone las cuatro entradas clave del hardware:
  - "x" y "y": Valores analógicos del joystick (e.g., 2048 central).
  - "sw" y "btn": Estado de los botones.
  - "connected": Bandera de conexión serial.

- **Manejo de Errores y Confiabilidad:** La función serial_worker incluye bucles de reconexión y manejo de excepciones (try-except). Si la conexión serial se

pierde (ser.close() se ejecuta o hay error), la bandera "connected" en el estado global se actualiza a False, cumpliendo parcialmente con la necesidad de tolerar desconexiones momentáneas.

6. Identificación de Requerimientos

Para desarrollar este juego, definimos los siguientes requerimientos:

No	Requisito del proyecto	Descripción	Tipo
RF01	Seleccionar modo de juego	El sistema permitirá jugar Jugador vs. Jugador y Jugador vs. IA (opcional).	Funcional
RF02	Navegación y selección de casillas	El mando ESP32 moverá un cursor por el tablero 3×3 y confirmará la casilla elegida. La GUI mostrará el estado en tiempo real.	Funcional
RF03	Pregunta antes de colocar la ficha	Antes de cada movimiento, la GUI mostrará una pregunta aleatoria del banco. Si la respuesta es correcta, se coloca la ficha; si es incorrecta, se pierde el turno.	Funcional
RF04	Validación de jugadas	El backend validará casillas libres, actualizará el tablero y detectará victoria/empate.	Funcional
RF05	Puntuación y tiempos	El sistema asignará puntuación extra por respuestas correctas rápidas.	Funcional
RF06	Temas educativos configurables	El usuario podrá elegir temas antes de iniciar.	Funcional
RF07	Gestión de banco de preguntas	El banco estará en JSON, con un mínimo de 10 preguntas por tema	Funcional
RF08	Estado y retroalimentación de HCI	La matriz de LEDs 3×3 reflejará selección y la ocupación de celdas; la GUI mostrará turno, pregunta y resultado.	Funcional
RF09	Inicio y fin de juego	Al conectar ESP32 y ejecutar el script Python, la GUI ofrecerá: seleccionar modo, tema, empezar partida y ver resultado final.	Funcional
RF10	Usabilidad	Flujo en menor a 3 pasos para iniciar partida; GUI legible y consistente	No Funcional
RF11	Rendimiento	Latencia entre acción en mando y reflejo en GUI y los LEDs en condiciones normales vía USB.	No Funcional
RF12	Confiabilidad	Tolerar desconexiones momentáneas de USB y descartar lecturas inválidas del ESP32.	No Funcional
RF13	Portabilidad	La GUI Python deberá ejecutarse al menos en Windows	No Funcional
RF14	Mantenibilidad	Código separado por capas: Dispositivo, Comunicación, Lógica de juego, GUI, Preguntas en JSON.	No Funcional

7. Implementación de Componentes

La implementación confirma la viabilidad técnica del proyecto, demostrando que la arquitectura Cliente-Servidor simple es efectiva para la integración de componentes heterogéneos (Hardware C/C++, Python/Flask, Web/JS).

Componente	Implementación Específica	Arquitectura
Generador de Preguntas	Función questions() en app.py junto con preguntas.json	Backend
Comunicación Serial	Hilo serial_worker usando la biblioteca serial y threading	Lector Serial
Motor de Interfaz	Vista principal de la página en index.html junto su comportamiento en javaScript game.js	Frontend
Cliente de Input	Función get_input() en app.py	Backend

8. Cronograma de actividades

Actividad	Encargado	Semana 1						Semana 2						Semana 3						Semana 4			
Definir dinámica base del juego	Jose Ángel, Giovani, Gabriel																						
Clasificación de requerimientos	Jose Ángel, Giovani, Gabriel																						
Establecer reglas del juego	Jose Ángel, Giovani, Gabriel																						
Elaborar lista de materiales	Jose Ángel, Giovani, Gabriel																						
Elaborar plan de trabajo	Jose Ángel, Giovani, Gabriel																						
Entrega de la primera fase	Jose Ángel, Giovani, Gabriel																						
Adquisición de materiales	Jose Ángel																						
Construcción del circuito del control	Jose Ángel																						
Cargar lógica de programación a la placa	Jose Ángel																						
Pruebas de conexión	Jose Ángel																						
Desarrollar diagrama de componentes	Jose Ángel, Giovani																						
Desarrollar interfaz del juego	Giovani																						
Desarrollar banco de preguntas	Gabriel																						
Integración del frontend con el banco de preguntas	Giovani, Gabriel																						
Integración de mandos del control con el juego	Jose Ángel, Gabriel																						
Pruebas de conexión	Jose Ángel, Gabriel																						
Elaboración de manual de usuario terminado	Jose Ángel, Giovani, Gabriel																						
Presentación en clase	Jose Ángel, Giovani, Gabriel																						

9. Código Fuente del Juego

9.1. Desarrollo del control

Para desarrollar el comportamiento del control utilizamos el siguiente código en nuestro microcontrolador ESP32, donde:

Definición de pines

- pinVRx = 34 y pinVRy = 35: entradas analógicas para los ejes horizontal y vertical del joystick.
- pinSW = 32: entrada digital para el botón integrado del joystick.
- pinBoton = 23: entrada digital para un pulsador externo.

Función setup():

- Serial.begin(115200);: inicia la comunicación serie a 115200 baudios.
- pinMode(pinSW, INPUT_PULLUP);: configura el botón del joystick como entrada con pull-up interno.
- pinMode(pinBoton, INPUT_PULLUP);: configura el pulsador externo igualmente con pull-up interno.

Función loop():

- analogRead(pinVRx);: lee la posición horizontal del joystick (x).
- analogRead(pinVRy);: lee la posición vertical del joystick (y).
- digitalRead(pinSW);: lee el estado del botón del joystick (sw).
- digitalRead(pinBoton);: lee el estado del pulsador externo (boton).

Formato de salida por Serial

- Imprime una línea con este formato: X:<valorX> Y:<valorY> SW:<1/0> BTN:<1/0>
- Para SW y BTN envía "1" cuando el botón está y "0" cuando está suelto.

Frecuencia de envío

- delay(50);: espera 50 ms al final del loop, de modo que se manda el estado del joystick y botones unas ~20 veces por segundo, suficiente para una lectura fluida en el juego.

```arduino
// Pines del joystick
const int pinVRx = 34;
const int pinVRy = 35;
const int pinSW = 32;

// Pin del pulsador
const int pinBoton = 23;

void setup() {
  Serial.begin(115200);
  pinMode(pinSW, INPUT_PULLUP);
  pinMode(pinBoton, INPUT_PULLUP);
}

void loop() {
  int x = analogRead(pinVRx);        // Movimiento horizontal
  int y = analogRead(pinVRy);        // Movimiento vertical
  int sw = digitalRead(pinSW);       // Botón del joystick
  int boton = digitalRead(pinBoton); // Pulsador externo

  Serial.print("X:");
  Serial.print(x);
  Serial.print(" Y:");
  Serial.print(y);
  Serial.print(" SW:");
  Serial.print(sw == LOW ? "1" : "0");
  Serial.print(" BTN:");
  Serial.println(boton == LOW ? "1" : "0");

  delay(50);  // Delay corto para lectura fluida
}
```

9.2. Interfaz

9.2.1. Backend del Juego

Para ello utilizamos la siguiente configuración para poder conectar los recursos como la dirección de archivo principal, las preguntas que usaremos para el juego y el puerto al cual conectamos los mandos del control a la interfaz.

```python
# Configuración
APP_DIR = os.path.dirname(os.path.abspath(__file__))
DATA_PATH = os.path.join(APP_DIR, "preguntas.json")
SERIAL_PORT = "COM3"  # <--- ¡VERIFICA TU PUERTO!
BAUD_RATE = 115200
```

Para obtener el estado del joystick usamos la siguiente configuración:

```python
# Estado global del Joystick
joystick_state = {"x": 2048, "y": 2048, "sw": 0, "btn": 0, "connected": False}
serial_lock = threading.Lock()
```

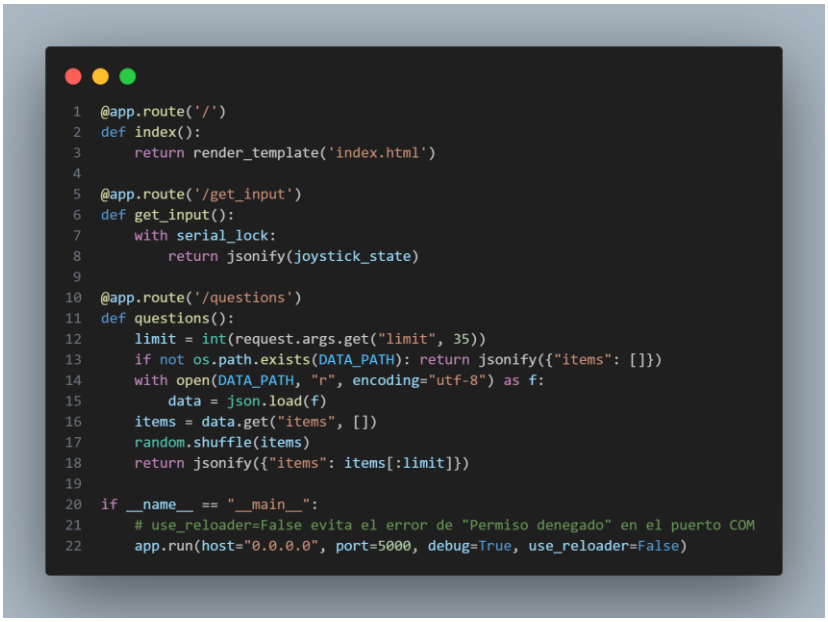
La función serial_worker() se encarga de mantener, en un hilo en segundo plano, la comunicación serie con el Arduino: intenta conectarse continuamente al puerto configurado, envía un comando inicial para encender un LED al lograr la conexión y, mientras la conexión esté activa, lee de forma constante las líneas de datos que el Arduino envía con las posiciones y estados del joystick (x, y, sw, btn); estos valores se guardan de manera segura en el diccionario global joystick_state usando un candado para evitar conflictos entre hilos, y en caso de producirse un error o desconexión, cierra el puerto, marca el joystick como no conectado y prepara el hilo para volver a intentar la conexión.

```python
def serial_worker():
    """Hilo que lee el Arduino en segundo plano"""
    global joystick_state
    ser = None

    while True:
        if ser is None:
            try:
                ser = serial.Serial(SERIAL_PORT, BAUD_RATE, timeout=0.1)
                print(f"✅ Conectado a {SERIAL_PORT}")
                time.sleep(2)
                ser.write(b'L') # Encender LED
                with serial_lock:
                    joystick_state["connected"] = True
            except Exception:
                with serial_lock:
                    joystick_state["connected"] = False
                time.sleep(2)
                continue

        try:
            if ser.in_waiting:
                line = ser.readline().decode('utf-8', errors='ignore').strip()
                if line.startswith("X:"):
                    parts = line.split()
                    with serial_lock:
                        joystick_state["x"] = int(parts[0].split(":")[1])
                        joystick_state["y"] = int(parts[1].split(":")[1])
                        joystick_state["sw"] = int(parts[2].split(":")[1])
                        joystick_state["btn"] = int(parts[3].split(":")[1])
                        joystick_state["connected"] = True
        except Exception as e:
            print(f"❌ Desconexión: {e}")
            try: ser.close()
            except: pass
            ser = None
            with serial_lock:
                joystick_state["connected"] = False

        time.sleep(0.01)
```

Este fragmento de código nos ayuda a comenzar los hilos iniciales para la conexión del juego.

```python
# Iniciar hilo serial
t = threading.Thread(target=serial_worker, daemon=True)
t.start()
```

Este fragmento define la parte principal de una aplicación web en Flask que sirve de interfaz entre el navegador, el joystick y un archivo de preguntas. La ruta raíz / devuelve la plantilla index.html, que es la página principal del cliente. La ruta /get_input expone, en formato JSON y de forma segura mediante el candado serial_lock, el estado actual del joystick almacenado en joystick_state, para que el front-end pueda leer sus valores en tiempo real. Por su parte, la ruta /questions lee un archivo JSON indicado por DATA_PATH, obtiene la lista de ítems (preguntas), los mezcla aleatoriamente, limita la cantidad según el parámetro limit recibido en la URL y regresa ese subconjunto como respuesta JSON. Finalmente, en el bloque if __name__ == "__main__": se inicia el servidor Flask escuchando en el puerto 5000 de todas las interfaces de red, con modo depuración activado y sin recargador automático para evitar conflictos con el puerto serie.



```python
@app.route('/')
def index():
    return render_template('index.html')

@app.route('/get_input')
def get_input():
    with serial_lock:
        return jsonify(joystick_state)

@app.route('/questions')
def questions():
    limit = int(request.args.get("limit", 35))
    if not os.path.exists(DATA_PATH): return jsonify({"items": []})
    with open(DATA_PATH, "r", encoding="utf-8") as f:
        data = json.load(f)
    items = data.get("items", [])
    random.shuffle(items)
    return jsonify({"items": items[:limit]})

if __name__ == "__main__":
    # use_reloader=False evita el error de "Permiso denegado" en el puerto COM
    app.run(host="0.0.0.0", port=5000, debug=True, use_reloader=False)
```

9.2.2. Frontend del Juego

Esta plantilla index.html define toda la interfaz visual del juego “SÚPER GATO”: carga las fuentes y la hoja de estilos, muestra videos de fondo distintos para el menú y para la partida, y organiza la página en varias pantallas (start-overlay, menu-screen y game-screen) que se ocultan o muestran según el estado del juego. Incluye el mensaje inicial de conexión con el joystick, el menú para elegir modo de juego (1 vs 1 o vs CPU), el tablero de tres por tres casillas con un cursor que se moverá sobre ellas, y un cuadro modal para mostrar avisos, mensajes y un temporizador. Además, define elementos de audio para la música y los efectos de sonido, y al final enlaza el archivo game.js, que será el responsable de añadir la lógica interactiva sobre esta estructura HTML.

```html
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>¡SÚPER GATO! 😾⚡</title>
    <link href="https://fonts.googleapis.com/css2?family=Titan+One&display=swap" rel="stylesheet">
    <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">
</head>
<body>

    <div id="background-container" style="background-color: #1a1a1a;">
        <video autoplay muted loop playsinline id="vid-menu" class="bg-video">
            <source src="{{ url_for('static', filename='assets/bg_menu.mp4') }}" type="video/mp4">
        </video>

        <video autoplay muted loop playsinline id="vid-game" class="bg-video hidden-video">
            <source src="{{ url_for('static', filename='assets/bg_game.mp4') }}" type="video/mp4">
        </video>

        <div class="overlay-pattern"></div>
    </div>

    <canvas id="confetti-canvas"></canvas>

    <div id="start-overlay">
        <div class="floating-content">
            <h1 class="mega-title">🎮 CONECTADO</h1>
            <p class="blink-text">Presiona el BOTÓN para entrar</p>
            <div id="debug-status" class="pill-status">Esperando señal...</div>
        </div>
    </div>

    <div id="menu-screen" class="screen hidden">
        <h1 class="game-logo">JUEGO DEL<br><span>GATO</span> 😾</h1>
        <div class="menu-container">
            <div id="btn-pvp" class="menu-btn selected">
                <span class="icon">🥊</span> 1 vs 1
            </div>
            <div id="btn-ia" class="menu-btn">
                <span class="icon">🤖</span> vs CPU
            </div>
        </div>
    </div>

    <div id="game-screen" class="screen hidden">
        <div class="game-ui">
            <div class="header-panel">
                <div id="turn-display" class="turn-badge">Turno: X</div>
            </div>
            <div id="board">
                <div class="cell" id="c0"></div><div class="cell" id="c1"></div><div class="cell" id="c2"></div>
                <div class="cell" id="c3"></div><div class="cell" id="c4"></div><div class="cell" id="c5"></div>
                <div class="cell" id="c6"></div><div class="cell" id="c7"></div><div class="cell" id="c8"></div>
                <div id="cursor"></div>
            </div>
        </div>
    </div>

    <div id="modal" class="modal hidden">
        <div class="modal-card pop-in">
            <div class="modal-header" id="modal-title">¡ATENCIÓN!</div>
            <div class="modal-body" id="modal-msg">Mensaje</div>
            <div id="modal-options" class="options-grid"></div>
            <div class="timer-wrapper hidden" id="timer-container">
                <div class="timer-icon">⏰</div>
                <div class="timer-track">
                    <div id="modal-timer-fill"></div>
                </div>
            </div>
        </div>
    </div>

    <audio id="music-menu" loop><source src="{{ url_for('static', filename='assets/menu.mp3') }}" type="audio/mpeg"></audio>
    <audio id="music-game" loop><source src="{{ url_for('static', filename='assets/game.mp3') }}" type="audio/mpeg"></audio>
    <audio id="sfx-place"><source src="{{ url_for('static', filename='assets/place.mp3') }}" type="audio/mpeg"></audio>
    <audio id="sfx-win"><source src="{{ url_for('static', filename='assets/win.mp3') }}" type="audio/mpeg"></audio>

    <script src="{{ url_for('static', filename='js/game.js') }}"></script>
</body>
</html>
```

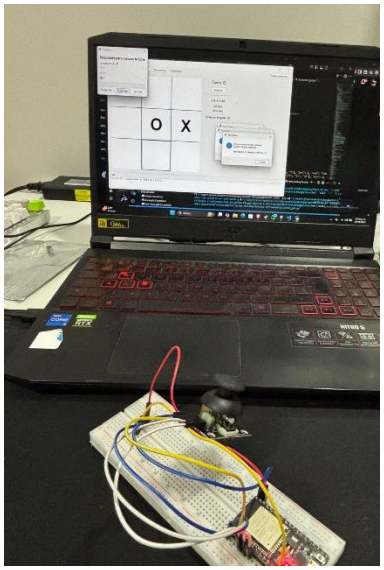
Para el comportamiento del juego lo realizamos a través de JavaScript donde a continuación se presenta una tabla con las funciones utilizadas en el juego:

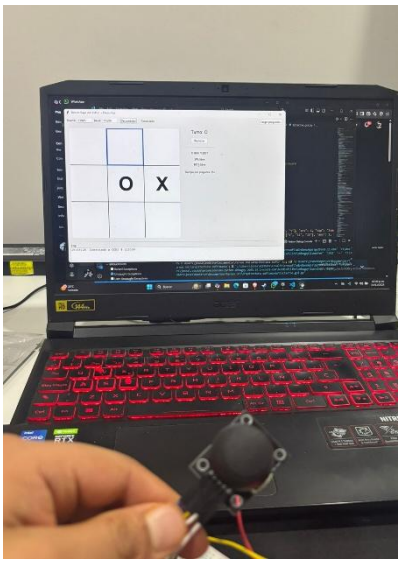
Función	Comportamiento en el juego
state (objeto global)	Mantiene todo el estado del juego: pantalla actual, tablero, turno, posición del cursor, modo de juego, temporizador, etc.
Objeto audio y volúmenes	Carga las pistas de música y efectos (menú, juego, colocar ficha, victoria) y ajusta sus volúmenes iniciales.
setInterval(.../get_input...)	Cada 50 ms consulta al backend el estado del joystick, actualiza el texto de conexión y llama a handleInput.
handleInput(data) – botón	Detecta si el botón del mando fue presionado (flanco de subida) y lo interpreta como clic para navegar o jugar.
handleInput(data) – start	En la pantalla de inicio, al primer botón desbloquea el audio, arranca la música del menú y llama a goMenu.
handleInput(data) – joystick	Traduce valores analógicos x e y del joystick a movimientos arriba/abajo/izquierda/derecha con anti-rebote.
handleInput según pantalla/modo	Redirige la entrada para controlar alertas, selector de preguntas, menú o movimiento en el tablero.
goMenu()	Cambia a la pantalla de menú, oculta overlay y juego, muestra el vídeo de menú y reproduce la música adecuada.
moveMenu(dy) / updateMenuVisuals()	Mueve la selección entre modos de juego con el joystick vertical y actualiza las clases CSS selected.
selectMenu()	Guarda el modo elegido (pvp o ia) y llama a startGame para iniciar la partida.
startGame()	Resetea tablero, turno y cursor, muestra la pantalla de juego, cambia videos y música, y dibuja el tablero.
moveCursor(dx, dy) / updateCursorPos()	Mueve el cursor por las 9 casillas respetando límites y reposiciona visualmente el div #cursor.
tryMove()	Si la casilla está libre, bloquea entrada y solicita una pregunta al backend antes de permitir colocar ficha.
showQuestion(q)	Muestra un modal con la pregunta y opciones, permite navegar con joystick y activa un temporizador.
startTimer()	Inicia una barra de tiempo de 10 s que se acorta cada 0.1 s; si llega a cero se pierde el turno.
moveQuestion(dy)	Cambia la opción seleccionada en el modal de pregunta según el movimiento vertical del joystick.
answerQuestion()	Comprueba si la opción elegida es correcta; si sí permite colocar ficha, si no pasa el turno.
placePiece()	Coloca la ficha en la casilla actual, reproduce sonido, anima la celda y evalúa victoria o empate.
checkWin()	Revisa todas las combinaciones ganadoras del gato y devuelve el símbolo ganador o false.
nextTurn()	Alterna el turno entre X y O, actualiza el texto del turno y, si corresponde, programa la jugada de la IA.

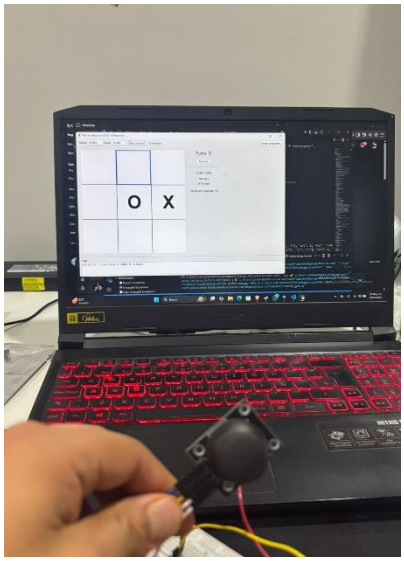
iaPlay()	Elige aleatoriamente una casilla libre para la IA, coloca O, evalúa victoria y devuelve el turno a X.
renderBoard()	Actualiza el contenido y estilo de cada celda del tablero según el valor almacenado (X, O o vacío).
showAlert(t, m, cb)	Muestra un modal de alerta con un único botón Aceptar y guarda un callback para ejecutar al cerrarlo.
closeModal()	Cierra el modal de alerta y ejecuta la función callback asociada.
launchExplosion()	Genera y anima partículas de confeti en un canvas para el efecto de victoria.
stopExplosion()	Detiene la animación de confeti y limpia el canvas.

10. Evidencias

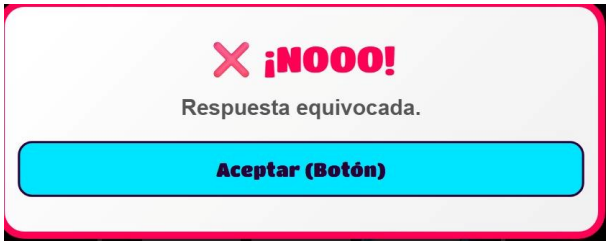
10.1. Desarrollo del control







10.2. Interfaz del Juego



11. Conclusión

Al analizar este código, queda claro que, aunque su estructura es relativamente corta, su función dentro del proyecto es fundamental. Sin este sketch, el joystick y el pulsador serían solo componentes de plástico y metal sin relación con lo que ocurre en pantalla. Gracias a la lectura constante de los ejes y botones, y al envío ordenado de los datos por el puerto serie, se consigue una interfaz de entrada confiable que permite al jugador controlar el juego del gato de una forma mucho más intuitiva y entretenida. Es decir, el programa convierte acciones tan simples como mover la palanca o presionar un botón en decisiones dentro del juego, ya sea desplazarse por un menú, elegir una casilla o validar una jugada.

Desde una perspectiva más técnica, el diseño resuelve varios puntos clave: se selecciona una velocidad de comunicación adecuada, se usa INPUT_PULLUP para garantizar lecturas estables de los botones y se define un formato de salida bien estructurado, fácil de parsear del lado del ordenador. Todo esto evita dolores de cabeza típicos como rebotes de señal, datos incompletos o mensajes difíciles de interpretar. Podría parecer que solo sirve para imprimir seriales, pero en realidad es una pequeña capa de protocolo que organiza la información y hace posible la sincronización entre el microcontrolador y el software de escritorio.

Al mismo tiempo, este código demuestra que no hace falta algo extremadamente complejo para lograr una experiencia de usuario divertida. Con unas cuantas líneas bien pensadas se puede pasar de un joystick desconectado a un control funcional que responde casi en tiempo real. A partir de esta base se pueden plantear muchas extensiones: calibrar los rangos del joystick, aplicar filtros para suavizar el movimiento, añadir más botones o incluso enviar información de vuelta al ESP32 para controlar LEDs, vibración u otros elementos de retroalimentación.

En conclusión, este programa es una pieza clave de la arquitectura general del proyecto: actúa como el intérprete entre el jugador y el sistema, mantiene una comunicación estable y prepara el terreno para que el resto del software el servidor en Python, la aplicación en Flask y el juego del gato en JavaScript puedan enfocarse en la lógica, la interfaz y la experiencia de juego. Es un buen ejemplo de cómo un módulo pequeño, bien diseñado y documentado, puede tener un impacto directo en la calidad y la sensación final de todo el proyecto interactivo.