Informe Proyecto Microprocesador Ajedrez

Gabriel Cascante Zamora* , Kamilah Chavarría Abarca* , Esteban Morera Ulate*, Daniel Urbina Siezar*

*Escuela de Ingeniería Mecatrónica, Instituto Tecnológico de Costa Rica (ITCR), 30101 Cartago, Costa Rica,
{gcascante, kamchavarria,jomorera,marurbina}@estudiantec.cr

***Resumen*—**El proyecto consiste en el diseño e implementación de un sistema capaz de jugar ajedrez, haciendo uso de un microprocesador y un microcontrolador. El problema principal es la necesidad de integrar la lógica completa del juego, con la interacción del usuario y la correcta representación física del tablero. Como solución se empleó una Raspberry Pi 5 para ejecutar la interfaz gráfica, procesar las entradas del usuario, validar las jugadas conforme a las reglas de ajedrez y generar movimiento lógicos en el modo automático. Toda la lógica del juego, incluyendo jugadas especiales como enroque, promoción y captura al paso, así como la detección de jaque mate y empate, se realiza en el microprocesador. Se hizo uso de un RobotDyn Mega basado en el ATmega2560 para la representación física del tablero mediante el uso de cuatro matrices LED 8x8, organizadas para formar un tablero completo, donde cada casilla es representada por 4 LEDSs con colores específicos para poder indicar el tipo de pieza y del jugador al que corresponde respectivamente. La comunicación entre ambos sistemas se hace mediante el protocolo de comunicación SPI, transmitiendo el estado completo del tablero cada vez que este se actualiza. Como resultado se logró realizar un sistema funcional capaz de recrear la jugabilidad del juego de ajedrez por completo, permitiendo operar tanto en modo asistido como automático y con su respectiva representación física.

***Palabras Clave*—** Ajedrez, Raspberry Pi 5, Microcontrolador, SPI, Interfaz gráfica, Multiprocessing.

I. Introducción

El proyecto de diseño planteado tuvo como objetivo general el uso de plataformas basadas en microprocesadores y microcontroladores, aprovechando recursos de software y hardware mediante la construcción de un sistema capaz de integrar esto a través del problema asignado, el cual consiste en implementar un sistema capaz de jugar ajedrez sobre un tablero estándar, integrando una interfaz gráfica en el microprocesador y un sistema basado en microcontrolador encargado de la representación física del tablero, comunicados mediante un protocolo definido.

Desde el punto de vista del software, el proyecto exige tres funcionalidades generales, las cuales se abordarán especificamente en la descripción del problema, como asignar/cargar un estado inicial del tablero, un modo asistente para dos usuarios, y un modo de juego donde el sistema realiza jugadas de forma autónoma. En el modo asistente, la interfaz debe presentar en todo momento la posición actual y marcar las jugadas con un código de color.

Para el modo autónomo no se espera que el algoritmo sea el más óptimo o inteligente, pero sí se espera que ejecute solo jugadas legales y que el sistema pueda reconocer una derrota bajo las condiciones del juego.

Conceptualmente, la solución requiere modelado de estado del tablero y sus transiciones (representación 8×8, turnos e historial), generación y validación de movimientos por pieza, incluyendo jugadas especiales (enroque, promoción y en passant), y paralelismo para separar tareas computacionalmente costosas del ciclo de interfaz, con el fin de mantener una ejecución fluida en la Raspberry Pi y evidenciar paralelismo además con el fin de aprovechar al máximo el microprocesador.

II. Descripción del problema resuelto

El problema a resolver consistió en desarrollar una aplicación de ajedrez funcional, ejecutable en una Raspberry Pi, que cumpliera con los requerimientos siguientes:

- **Funcionalidad:** representar el tablero 8×8, permitir el salto de turno, registrar el historial de jugadas y permitir aplicar y deshacer movimientos manteniendo consistencia del estado.
- **Generación y validación de movimientos:** calcular movimientos por pieza y filtrar aquellos que no son legales, es decir, jugadas que dejan al propio rey en jaque. El sistema debe identificar correctamente estados terminales del juego.
- **Detección de fin de juego:** determinar jaque mate cuando el jugador en turno está en jaque y no tiene movimientos legales, y stalemate empate cuando no hay movimientos legales pero el rey no está en jaque.
- **Jugadas especiales:** implementar enroque (con control de derechos y verificación de casillas atacadas), en passant (con seguimiento de la casilla capturable) y promoción de peón (con selección de la pieza resultante).
- **Interfaz gráfica y flujo de uso:** brindar una GUI interactiva para seleccionar piezas y destinos, con pantallas de menú y setup, permitiendo iniciar una partida nueva o cargar una posición inicial.
- **Asistencia visual al jugador:** mostrar indicaciones de jugadas con códigos de color para apoyar la toma de decisiones: movimientos legales a casillas vacías (verde), capturas legales (amarillo) y movimientos inválidos (rojo), además de mensajes de error cuando se intente una jugada ilegal.

- **Modo automático (IA):** incluir un modo donde el sistema realice jugadas legales de forma autónoma, evaluando opciones mediante una estrategia básica.
- **Paralelismo en el microprocesador:** evidenciar ejecución paralela separando tareas más costosas como el cálculo de movimientos legales y selección de jugadas de la IA del hilo principal de la interfaz, manteniendo fluidez durante la ejecución.
- **Integración con microcontrolador:** contemplar la comunicación con un sistema basado en microcontrolador mediante un protocolo definido, para enviar el estado del tablero o señales de eventos del juego y que esta sea interpretada y ejecutada mediante interrupciones cuando se requiere para realizar una representación física en una maqueta.

III. Enfoque e implementación de la solución

La solución se diseñó con un enfoque modular, separando responsabilidades en scripts independientes para facilitar pruebas, debugging y extensibilidad. A nivel de software, se definieron cinco módulos principales: motor de reglas y estado del juego (Engine.py), interfaz gráfica y control de flujo (Main_Setup.py), inteligencia artificial para el modo automático (IA.py), proceso trabajador para cálculos pesados en paralelo (mp_worker.py) y comunicación SPI hacia microcontrolador (spi_link.py). La figura 1 resume la arquitectura general.

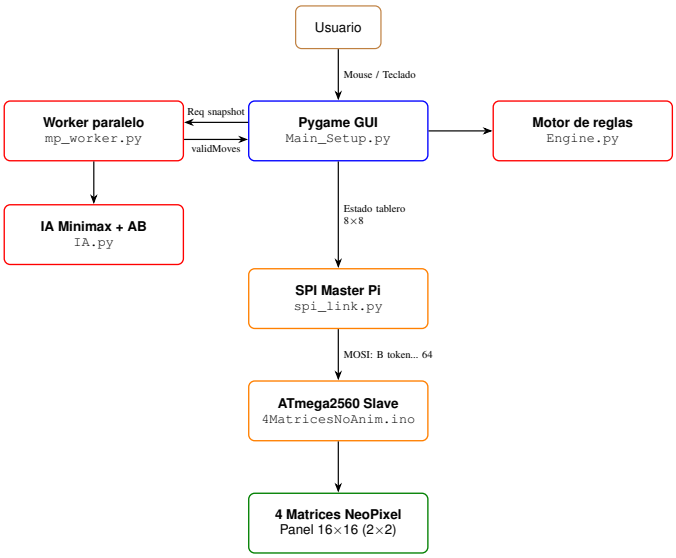

Figura 1: Arquitectura del sistema: Software, Comunicación y Hardware.

III-A. Arquitectura general del sistema físico

La arquitectura funciona de manera que la GUI coordina entradas/salidas y delega la lógica a una máquina de estados que define el estado actual del programa. Para cumplir con el paralelismo, se usa un segundo proceso que replica el estado del tablero mediante snapshots (tablero, turno, en passant y derechos de enroque) y calcula: movimientos válidos, y la mejor jugada de la IA. En la figura 2 se encuentra un diagrama de bloques representando el hardware.

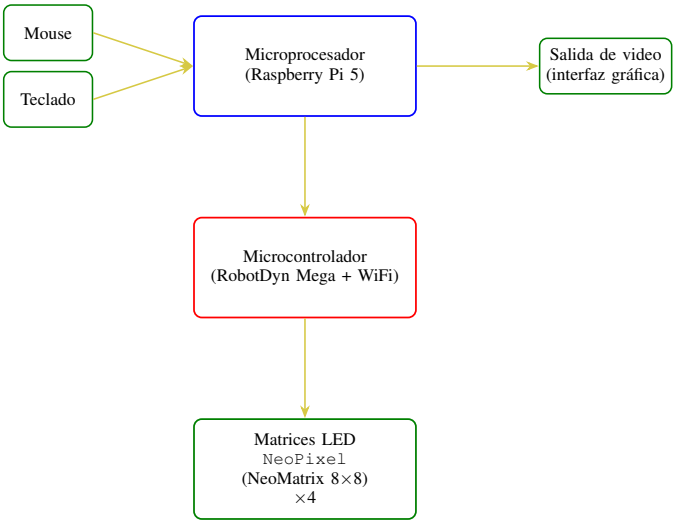

Figura 2: Diagrama de bloques del hardware.

III-B. Flujo de ejecución

La aplicación se organiza como una máquina de estados finitos (FSM) con tres pantallas: Menu, Setup y Game. El ciclo principal procesa eventos de Pygame (mouse/teclado), actualiza el estado y renderiza la pantalla correspondiente. En Game, el cálculo de movimientos legales y la selección de jugada de IA se solicitan al worker para no bloquear la interfaz. A continuación en la figura 3 se presenta un diagrama de flujo de la funcionalidad general completa del software empleado para implementar la lógica del ajedrez en el microprocesador para ser mostrado en la interfaz gráfica.

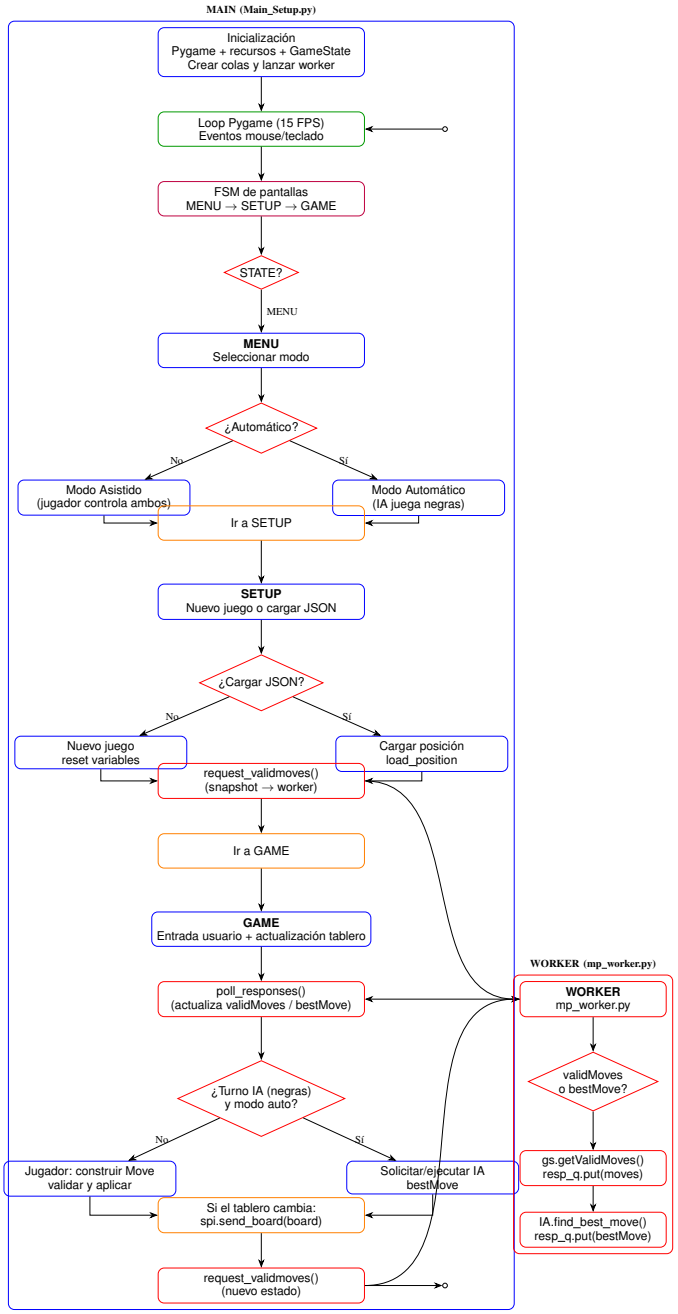

Figura 3: Flujo general del software: selección de modo (asistido/automático), configuración (nuevo/JSON) y ejecución en GAME con cálculo asíncrono (worker) y sincronización por SPI.

III-C. Implementación del motor de ajedrez (Engine.py)

El motor se centró en la clase GameState, que mantiene:

- **Tablero:** matriz 8×8 con codificación de piezas.
- **Turno e historial:** El turno se guarda en la variable whiteToMove, donde cuando esta es 1 el turno es de blancas y cuando es 0 es turno de negras y un log de todas las jugadas llamado moveLog.
- **Reyes:** ubicaciones de ambos reyes para verificar jaques.
- **Jugadas especiales:** Inclusión de enroque, En passant y promoción al detectar una determinada secuencia de movimientos.

Se implementaron funciones por pieza (getPawnMoves, getRookMoves, etc.) para generar movimientos posibles. Luego, la función llamada getValidMoves() filtra jugadas legales simulando cada movimiento, verificando jaque con inCheck(), y deshaciendo con la función undoMove(). Este método prioriza corrección y simplicidad, evitando errores al manejar jugadas especiales y el estado del rey. Haciendo que la generación y validación de movimientos legales sigue principios clásicos de búsqueda en juegos determinísticos [1].

III-D. Implementación de la interfaz y estados (Main_Setup.py)

La GUI se desarrolló en Pygame y se organizó con estados:

- **MENU:** selección de modo (Asistido / Automático).
- **SETUP:** inicio de partida nueva o carga de posición desde .json.
- **GAME:** tablero interactivo, mensajes y visualización de jugadas.

III-E. Implementación de IA para modo automático (IA.py)

El módulo IA.py implementa el modo automático usando Minimax con poda Alpha–Beta a una profundidad configurable. La idea es evaluar varias jugadas legales hacia adelante y escoger la que produzca el mejor resultado según una función de puntaje simple. No se busca una IA optima, sino un comportamiento coherente que siempre juegue movimientos legales.

Para puntuar una posición se usa una evaluación por material: se asignan valores a las piezas según la clase (peón, caballo, alfil, torre y reina). El puntaje se calcula recorriendo el tablero, sumando el valor de piezas blancas y restando el de piezas negras, por eso, puntajes positivos favorecen a blancas y negativos favorecen a negras. En paralelo, se manejan estados terminales con constantes como checkmate score (muy alto) para forzar que la IA prefiera ganar por mate o evite perder por mate, y stalemate score (cero) para representar un empate. La función terminal score detecta estos casos verificando si no hay movimientos legales y si el rey del jugador en turno está en jaque.

Un detalle importante es que getValidMoves() del motor puede modificar banderas como checkMate y staleMate. Para evitar que esas banderas queden alteradas durante simulaciones internas de la IA, se usa _safe_valid_moves(gs), que llama a getValidMoves() pero guarda y restaura las banderas. La búsqueda en _minimax funciona haciendo y deshaciendo jugadas, además de actualizar los límites alpha y beta, los cuales son utilizados para ”podar ramas”, reduciendo el tiempo de cálculo.

La función find best move es la interfaz principal, pues esta recibe la lista de movimientos legales actuales, los baraja con shuffle para romper empates y evitar patrones repetitivos, y luego evalúa cada movimiento con Minimax para escoger el mejor. Si una jugada implica promoción, la IA prueba

explícitamente las cuatro opciones posibles y selecciona la que produzca el mejor puntaje en la búsqueda, guardando esa elección.

III-F. Implementación de multiprocessing con mp_worker.py

Se implementó paralelismo con colas y un proceso worker. El *worker*:

- reconstruye un *snapshot* del estado mediante load_position(),
- aplica variables necesarias para exactitud en passant y derechos de enroque,
- calcula movimientos legales, y opcionalmente calcula la mejor jugada de la IA.

El proceso principal se mantiene responsivo porque renderiza a 15 FPS mientras recibe resultados asíncronos. Para evitar usar respuestas obsoletas, se emplea un position_id que permite aceptar solo la respuesta más reciente solicitada. La separación entre tareas de interfaz y cómputo intensivo es una práctica recomendada para mantener sistemas interactivos responsivos [2].

III-G. Comunicación SPI con spi_link.py

La integración entre el microprocesador (Raspberry Pi) y el microcontrolador se planteó mediante el protocolo **SPI** (*Serial Peripheral Interface*), cuyas funciones las contiene el script spi_link.py a través de la clase SpiLink. En este diseño la Raspberry Pi opera como master y el microcontrolador como slave, de forma que el microprocesador controla el reloj de comunicación y el momento de la transferencia mediante la línea de selección. Esta elección simplifica la sincronización y permite que el microcontrolador reciba actualizaciones del juego de manera inmediata y determinista.

El estado del tablero se transmite desde la Raspberry Pi como un mensaje de texto con un encabezado `"B"` seguido por 64 tokens que representan las 64 casillas en orden fila–columna, utilizando la codificación del motor del juego. Además, se incluyen funciones auxiliares, las cuales permiten extender el protocolo para notificar eventos del juego, como jaque mate o empate.

SPI se seleccionó frente a alternativas como I2C o RS-232 por razones prácticas relacionadas con el tipo de información a transmitir y el comportamiento esperado del sistema. En comparación con I2C, SPI ofrece mayor velocidad y un esquema de comunicación más directo para un enlace punto a punto (master y slave), sin necesidad de direcciones ni arbitraje del bus. Esto es conveniente cuando se desean actualizaciones frecuentes del estado del tablero y baja latencia en la recepción. Por otro lado, frente a RS-232 (UART), SPI tiene la ventaja de ser síncrono (con reloj), lo cual reduce incertidumbre de temporización y facilita mantener transferencias consistentes cuando el microprocesador controla el ritmo de envío. RS-232 resulta útil para enlaces de depuración o distancias mayores, pero para una conexión corta dentro del sistema y con requerimiento de actualizaciones rápidas, SPI es una alternativa más simple y eficiente [3].

III-H. Implementación física: despliegue del tablero con 4 matrices LED 8×8

Para la implementación física se construyó un sistema de visualización del tablero basado en cuatro matrices LED de 8×8 conectadas en cadena, para un total de 256 LEDs. Estas matrices se organizaron como un panel de 16×16, de manera que el tablero real de ajedrez (8×8) pudiera representarse de forma clara asignando cada casilla a un bloque de 2×2 LEDs. Esta decisión permite aumentar la "resolución" por casilla, mejorar la visibilidad a distancia y facilitar la diferenciación de piezas mediante color, sin necesidad de un display más complejo.

El microcontrolador RobotDyn Mega es el dispositivo de despliegue y recibe desde el microprocesador el estado completo del tablero mediante SPI. En el firmware, la recepción SPI se realiza en interrupción, almacenando byte a byte en un buffer hasta detectar el salto de línea.

Una vez recibido un mensaje válido (encabezado `"B"` seguido de los 64 tokens, el programa limpia el panel y recorre los elementos en orden fila–columna. Para cada token se calcula su posición en el tablero y se convierte a un color mediante la función pieceToColor(), que define una paleta distinta para piezas blancas y negras. Posteriormente, se encienden los 4 LEDs correspondientes a esa casilla (bloque 2×2). Finalmente, cuando se procesan las 64 casillas, se actualiza el despliegue con strip.show().

Debido a que las matrices están conectadas en cadena y pueden estar montadas con distintas orientaciones fíisicas, el firmware incluye una capa de mapeo para garantizar que el tablero se despliegue correctamente sin tener que re-cablear. En primer lugar se incorporan banderas globales FLIP_X y FLIP_Y para corregir inversiones izquierda–derecha o arriba–abajo del tablero completo. Además, se define el orden de los cuadrantes del panel para indicar cuál de las cuatro matrices físicas corresponde a cada cuadrante del panel. Finalmente, también se permite especificar la orientación individual de cada matriz (normal, flip horizontal, flip vertical o rotación 180°). Con estas configuraciones, se traducen coordenadas del panel a un índice global de LED (0 a 255), aplicando el cuadrante correcto y la orientación apropiada.

Se configuró un brillo bajo debido a que 256 LEDs pueden demandar una corriente considerable a brillo máximo. Esto se incluyó como medida de seguridad y estabilidad del sistema durante pruebas continuas. La figura 4 muestra la conexión física entre el microprocesador y el microcontrolador para SPI, así como la conexión del pin de datos hacia la entrada de la cadena de matrices LED y la alimentación apropiada del panel.

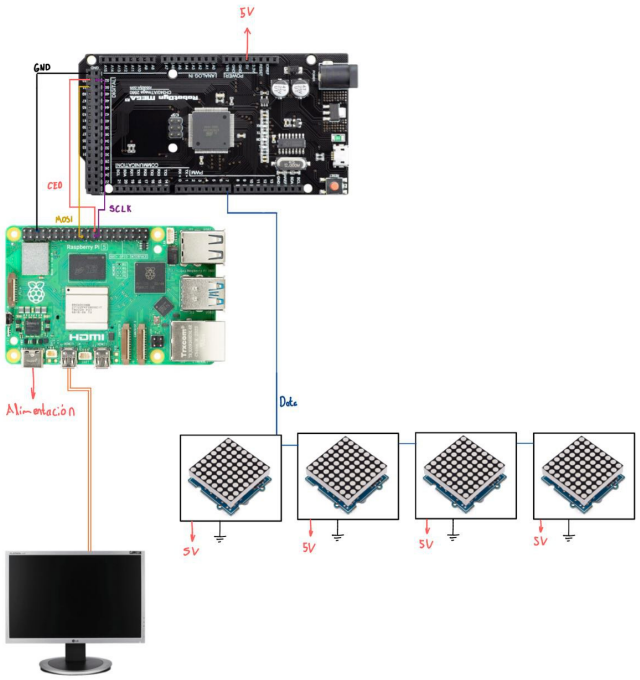

Figura 4: Esquema de conexión física entre el microprocesador, el microcontrolador y las matrices LED 8×8.

IV. Análisis de la solución

Para cada una de las subdivisiones principales del proyecto se va a realizar un análsis para determinar su funcionalidad dentro del proyecto y el cumplimiento de los requerimientos establecidos, así como valorar su rendimiento y explicar la toma de decisiones respecto al diseño establecido.

IV-A. Distribución general del hardware

La solución se diseñó para seguir los lineamientos impuestos por el enunciado al mismo tiempo que aprovechar las fortalezas de cada plataforma. La Raspberry Pi 5 ejecuta la interfaz en Pygame, la máquina de estados de la aplicación y el motor lógico del ajedrez, todas estas tareas requieren de una mayor memoria, facilidad de manejo de archivos y poder computacional para hacer el cálculo de validación y jugadas de IA, aprovechando funcionalidades como el multiprocessing que el microcontrolador no ofrece. El RobotDyn Mega se encarga de todo el despliegue físico con los timings específicos que requiere Neopixel para hacer la representación visual, aprovechando las interrupciones para comunicación. Esta separación permite depurar independientemente la lógica del juego y el funcionamiento de la representación física, al mismo tiempo que aprovechar las ventajas específicas que tiene cada sistema.

IV-B. Diseño de interfaz gráfica

La interfaz se esctructura como una máquina de estados claramente definida, que separa las pantallas de menú, configuración inicial y juego activo. Esta organización se decidió para evitar mezclar la lógica de presentación con las reglas del juego, lo que permitió mejorar la claridad del flujo del programa y facilitar debuggearlo.

El sistema soporta distintos modos de operación:

- selección del modo asistido o automático
- carga de tableros iniciales personalizados desde el explorador de archivos
- interacción por mouse y teclado para selección de piezas, cambio de turno y salida del programa

En el modo de juego, el sistema resalta visualmente jugadas legales, ilegales y capturas, e incorpora un mecanismo de advertencia ante intentos de movimientos inválidos. Esto mejora la experiencia del usuario y reduce errores durante la interacción

A continuación en la figura 5 se encuentran las tres pantallas que se pueden obtener en la interfaz gráfica.

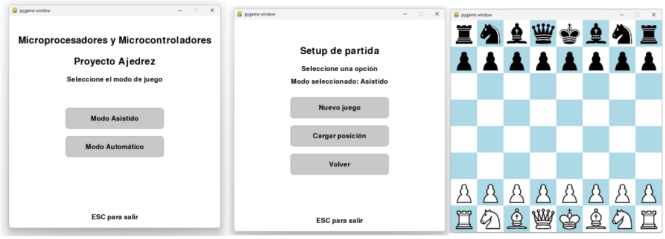

Figura 5: Pantallas de estados de la interfaz gráfica

IV-C. Multiprocessing

Uno de los principales retos del proyecto es el costo computacional asociado al cálculo de movimientos legales y a la evaluación de la inteligencia artificial para el modo automático. Para evitar que la interfaz gráfica pierda fluidez durante estos cálculos, se implementó paralelismo mediante multiprocessing. El sistema utiliza un proceso secundario (worker) encargado de:

1. Calcular la lista de movimientos válidos para un estado dado.
2. Determinar la mejor jugada en modo automático mediante el algoritmo de IA.

El proceso principal le envía al worker un snapshot del estado actual del tablero (matriz, turno, derechos de enroque y en passant), mientras mantiene la actualización de la interfaz gráfica a 15 FPS y la recepción de eventos del usuario. La comunicación entre procesos se realiza mediante colas (Queue). Este diseño permite que la interfaz se mantenga responsiva incluso mientras la IA evalúa posiciones complejas, cumpliendo con el requisito de uso de paralelismo.

En la figura 6 se muestra lo que se imprime en terminal al ejecutar el worker, validando su funcionalidad.

```
pygame 2.6.1 (SDL 2.28.4, Python 3.11.9)
Hello from the pygame community. https://www.pygame.org/contribute.html
worker.pid = 17356
worker.is_alive() = True
worker.exitcode = None
pygame 2.6.1 (SDL 2.28.4, Python 3.11.9)
Hello from the pygame community. https://www.pygame.org/contribute.html
```

Figura 6: Activación de worker para multiprocessing

IV-D. Inteligencia artificial y criterios de decisión

A pesar de ser un extra del proyecto, ya que con simplemente implementar un random dentro de la lista de `validMoves` se podía obtener un modo automático, se decidió implementar una inteligencia artificial basada en el algoritmo Minimax con poda Alpha-Beta para mejorar la experiencia del modo de juego automático.

Esta utiliza una función de evaluación simple basada en el valor del material de las piezas. Este enfoque le permite al sistema proporcionar un nivel de dificultad media-alta suficiente para generar partidas coherentes sin comprometer en exceso los tiempos de cálculo.

Durante las promociones el algoritmo evalúa todas las opciones posibles (Reina, torre, alfil y caballo) y selecciona la que maximiza la evaluación del tablero.

Al emplear este algoritmo se logra tener un modo automático responsivo en que se castigan las malas jugadas, al tener la capacidad de buscar piezas por comer y premiar el jaque mate, dando la sensación de una partida real.

IV-E. Comunicación SPI

Finalmente como último sistema relevante del proyecto se tiene la comunicación con el microcontrolador y la representación física del tablero a través de la actualización por medio del protocolo de comunicación SPI.

Esta se implementa utilizando la Raspberry Pi como maestro y el RobotDyn Mega como esclavo. Esta elección se fundamenta en la simplicidad del protocolo, su baja latencia y la facilidad de sincronización, además de permitir el uso de interrupciones en el microcontrolador.

Además, se prioriza un esquema de comunicación de una sola dirección de la Raspberry Pi 5 hacia el microcontrolador, debido a que el RobotDyn Mega podría producir señales de 5V que pongan en riesgo al microprocesador al trabajar con 3.3V.

El estado del tablero se transmite como un mensaje que contiene un identificador de inicio y final de mensaje y los 64 valores correspondientes de las casillas. Del lado del microcontrolador la recepción se maneja mediante la interrupción `SPI_STC_vect`, acumulando los bytes hasta comprobar una línea válida. Esto permite una actualización determinista del tablero físico y cumple con el requisito de uso de interrupciones.

La representación física utiliza una malla 16×16 de LEDs NeoPixel organizada en cuatro matrices 8×8, asignando 2×2 LEDs por casilla. Esta solución mejora la visibilidad del tablero y simplifica el mapeo entre el modelo lógico y el hardware.

V. Conclusiones

El desarrollo de este proyecto permitió implementar exitosamente un sistema de ajedrez basado en una arquitectura distribuida, cumpliendo en su totalidad los requerimientos funcionales y técnicos establecidos. La separación de funciones entre la Raspberry Pi 5 y el microcontrolador ATmega2560 demostró ser una decisión adecuada, ya que permitió asignar tareas complejas de interfaz, lógica y cálculo a la plataforma con mayor capacidad de procesamiento, mientras que el microcontrolador se encargó de la representación física del tablero de forma eficiente.

La lógica del ajedrez implementada permitió manejar correctamente las reglas del juego, incluyendo jugadas especiales como enroque, promoción y captura al paso, así como la detección de jaquemate y stalemate. El uso de paralelismo mediante multiprocessing permitió mantener una interfaz gráfica fluida durante el cálculo de movimientos legales y del algoritmo autónomo, dejando en evidencia la importancia de separar las tareas de mayor nivel de computación de la interacción del usuario.

Finalmente, la comunicación mediante SPI y el uso de interrupciones en el microcontrolador para recepción de mensajes permitieron sincronizar el estado del tablero con su representación física.

VI. Recomendaciones

Se recomienda como trabajo futuro, ampliar el algoritmo del modo autónomo, incorporando criterios posicionales, como control del centro, estructura de peones y demás, mejorando la estrategia de jugadas sin aumentar la profundidad de búsqueda.

Finalmente, se recomienda extender la interfaz gráfica y su representación física, incluir animaciones en la matriz de LEDs para jaquemate y stalemate, indicadores visuales de jugadas especiales, marcadores de jugada previamente realizada y demás.

Referencias

[1] Stuart Russell and Peter Norvig, *Artificial Intelligence: A Modern Approach*. Pearson, USA.

[2] Michael McCool and Arch Robison and James Reinders, *Structured Parallel Programming*. Morgan Kaufmann, USA.

[3] Total Phase, *I2C vs SPI vs UART – Introduction and Comparison of their Similarities and Differences*. Total Phase, USA.