IAChess

Jorge Pérez Seco

12/2024

1 Explicación del Objetivo

En este proyecto, el objetivo principal ha sido desarrollar una inteligencia artificial (IA) de ajedrez. Para hacer este trabajo original, el programa debe proporcionar las siguientes funcionalidades:

- La opción de jugar contra la IA con piezas blancas o negras.
- La posibilidad de iniciar desde una posición concreta, especificada en formato FEN.
- Un menú que permita al usuario ejecutar diversas acciones, entre las cuales se incluyen:
  - Retroceder movimientos.
  - Obligar a la IA a cambiar su movimiento.
  - Salir del programa.
  - Realizar un movimiento en notación algebraica estándar (SAN).

El objetivo del programa no es analizar partidas, sino proporcionar un bot que juegue contra el usuario. Esto tiene implicaciones específicas en el diseño del código, que se describirán más adelante. La parte más relevante en cuanto al concepto de inteligencia artificial radica en el cálculo del mejor movimiento.

El programa calcula los primeros X movimientos de una línea (secuencia de movimientos derivados de otro). Posteriormente, la red neuronal, en

principio preentrenada, selecciona los mejores movimientos. Con esta información, el programa aplica el algoritmo *minimax* con una profundidad Y, donde $Y > X$, para determinar el mejor movimiento basado en la valoración material al final de dichas líneas. Este proceso será explicado en mayor detalle en la descripción del proceso.

2 Descripción del Proceso

El desarrollo de este proyecto comenzó con un primer esbozo generado con la ayuda de ChatGPT. En esta etapa inicial, el programa consistía en una inteligencia artificial (IA) no entrenada que jugaba partidas de ajedrez contra el usuario. Este prototipo ya incluía funcionalidades como la posibilidad de introducir una posición inicial en formato FEN y la opción de jugar con ambos colores. Sin embargo, carecía de un menú de comandos y tenía problemas para interpretar correctamente la notación algebraica estándar (SAN).

Para abordar estas limitaciones, implementé un menú de comandos con la ayuda de OpenAI. Este menú permite al usuario realizar diversas acciones, como deshacer movimientos utilizando una lista para almacenar el historial de jugadas y aplicar el método `pop()`. Finalmente, decidí no sustituir esta estructura por un *heap*, ya que mi objetivo no era crear una herramienta de análisis de partidas, sino un bot para jugar contra el usuario. Es importante mencionar que, en el caso de un sistema de análisis de partidas, un *heap* sería más adecuado, pues permitiría navegar entre distintas líneas de juego.

Además, al implementar el comando para reiniciar el tablero, tuve que asegurarme de que también se eliminara el historial de movimientos, ya que de no hacerlo, los movimientos se seguirían aplicando sobre una matriz desactualizada. También eliminé elementos superfluos en el código, como en la línea 56 del prototipo inicial, y realicé ajustes adicionales para que todos los comandos del menú funcionaran correctamente.

En cuanto al problema de la notación SAN, inicialmente el programa daba la opción de utilizar notación UCI. Sin embargo, debido a una implementación defectuosa de un condicional, el programa interpretaba incorrectamente cualquier cadena de cuatro caracteres como notación UCI, lo que causaba conflictos con la notación SAN. Para resolver este problema, eliminé por completo la compatibilidad con la notación UCI, ya que la SAN es el estándar ampliamente utilizado por los jugadores.

La IA no está entrenada debido a limitaciones de tiempo. Sin embargo, al

probar el programa, noté que la evaluación de las posiciones no era realista en comparación con el tablero mostrado. Esto se debe a que, a diferencia de las *engines* tradicionales que evalúan posiciones asignando valores numéricos a las piezas según su utilidad (positivos para blancas y negativos para negras), mi programa inicial confiaba completamente en la red neuronal para valorar el peso posicional de las piezas. Además, el programa consideraba únicamente el valor material de las piezas.

Aunque esta perspectiva puede resultar interesante en una IA entrenada, actualmente la IA tiene un enfoque demasiado materialista. Dado que las profundidades de búsqueda X e Y no superan los 6 movimientos para evitar tiempos excesivos de cálculo, su nivel de juego es similar al de un jugador con un ELO de 700.

Para explorar un enfoque diferente, desarrollé un código alternativo llamado `IAChess2.0.py`, que incorpora la idea de usar coeficientes relativos al valor práctico de las casillas en el tablero. Este programa asigna valores a las piezas y ajusta dichos valores según la casilla en la que se encuentran. El nuevo enfoque permite una valoración más efectiva del tablero y muestra un desempeño más lógico en las partidas. A continuación, presento un ejemplo de partida generada por este segundo código:

```
1. d4 Nf6
2. Qd3 e6
3. Bf4 c5
4. Nc3 Nc6
5. O-O-O cxd4
6. Ne4 Nxe4
7. Qxe4 d5
8. Qd3 Qa5
9. Nf3 Qxa2
10. Rd2 Qa1# 0-1
```

En esta partida, las blancas abren con `d4`, controlando el centro del tablero. Sin embargo, debido a la limitada profundidad de cálculo, la IA muestra una preferencia excesiva por llevar piezas valiosas al centro, lo que resulta en movimientos subóptimos como `Qd3`. Aunque la partida no dura mucho más, los movimientos realizados son lógicos considerando la programación y las limitaciones del modelo.

Para visualizar partidas o realizar análisis, se puede utilizar el enlace `https://lichess.org/paste`, donde es posible pegar los movimientos en

formato SAN. Por ejemplo, la partida anterior puede ser analizada utilizando este enlace.