Grado en Ingeniería Informática

Curso 2016-2017

Trabajo Fin de Grado

"Búsqueda Automática de Problemas de Táctica en Partidas de Ajedrez"

Lauro B. Bravar Abril-Martorell

Tutor

Profesor Carlos Linares López

10 de octubre 2017

Agradecimientos

A mi tutor, el Profesor Carlos Linares, por ser mi guía en este proyecto y por ayudarme y motivarme durante toda la carrera.

A mis padres y padrastro, sin ellos nada de esto sería posible.

A mi abuelo Eligio, por ser la causa principal de mi amor por el ajedrez.

A Lucía, por aguantar mis penas y alegrías durante el transcurso de este proyecto.

A los grandes maestros y estudiosos de este maravilloso arte y juego, pues todos ellos son en parte creadores de este trabajo.

Índice

Índice Ilustraciones

1. Introducción

1.1 Motivación

Tanto el tutor de este proyecto, el Profesor Carlos Linares, como el autor del mismo compartimos una enorme afición por el ajedrez. Ambos hemos jugado y amado este juego desde que éramos críos y es por ello que el profesor me propuso la realización de este trabajo que con gusto acepté.

El proyecto surge de la importancia que tiene para un jugador de ajedrez aprender de los errores que comete. Como dijo la leyenda del ajedrez José Raúl Capablanca:[1] "*De pocas partidas he aprendido tanto como de la mayoría de mis derrotas*".

Los problemas de táctica son sin duda una buena forma de mejorar la habilidad de un jugador de ajedrez, especialmente en este aspecto del juego, la táctica. Pero para entender qué es la táctica y qué significa en el ajedrez es necesario saber qué es y cómo se juega al ajedrez.

[1] José Raúl Capablanca (La habana, Cuba, 1888- Nueva York, US, 1942). Ajedrecista cubano campeón mundial de ajedrez desde 1921 a 1927. Referente de ajedrez a nivel mundial.

1.2 Estructura del documento

El documento se divide en ocho partes:

- **1. Introducción**

  En esta primera parte del trabajo se describe brevemente el problema a tratar, la estructura del documento, la motivación que hay detrás del mismo y una explicación de lo que es el ajedrez.

- **2. Estado del arte**

  La segunda parte trata el estado de la cuestión a tratar. En este se incluye la historia del ajedrez, la definición de táctica y problema de táctica, y cómo se ha abordado este problema computacionalmente.

- **3. Algoritmos de búsqueda en el ajedrez**

  En este apartado se tratan los algoritmos Minimax y Alfa Beta, los algoritmos de búsqueda más utilizados en el ajedrez.

- **4. Solución**

  La cuarta parte del documento expone la solución del problema. Cómo se ha desarrollado, estructura del código y elementos que lo componen así como las posibles mejoras futuras sobre el programa.

- **5. Pruebas**

  La quinta parte expone las pruebas llevadas a cabo con la solución desarrollada y los resultados obtenidos con estas pruebas.

- **6. Marco regulador**

  En la sexta parte se muestra el marco legal del proyecto.

- **7. Cronograma y presupuesto económico**

  En esta parte se expone tanto el cronograma seguido durante el proyecto como el presupuesto económico del mismo.

- **8. Conclusión**

  Por último, en la octava parte del documento, se exponen brevemente las conclusiones obtenidas tras la realización del proyecto.

1.3 Ajedrez

El ajedrez es un juego de mesa, para algunos incluso un arte, para dos personas (si no incluimos las variantes de *bughouse*[2] y *ajedrez para 3*[3]).

El juego lo componen un tablero y 32 piezas y, en juegos regulados, un reloj. El tablero está formado por 64 casillas repartidas en una cuadrícula de 8 por 8. Las casillas pueden ser blancas o negras, y los colores se distribuyen de forma alterna empezando por una casilla de color negro en la esquina inferior izquierda del tablero desde la posición de las blancas como se muestra en la siguiente imagen:

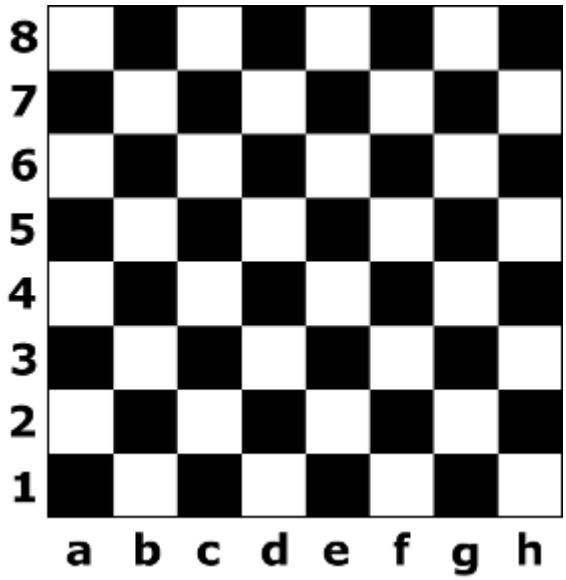

Ilustración 1 - Tablero de ajedrez vacío

Fuente: elaboración propia

[2] Modalidad de juego del ajedrez en el que juegan 4 jugadores, dos contra dos. las piezas que se toman en un tablero se pueden colocar utilizando un turno en el otro tablero.

[3] Modalidad de juego del ajedrez en la que el tablero ha sido modificado para que jueguen tres jugadores, todos contra todos.

Las letras y números que aparecen a los lados del tablero corresponden con la nomenclatura que recibe cada casilla. Siendo la casilla inferior izquierda la casilla *a1* y la superior derecha la *h8*. Esta nomenclatura es útil, como se mostrará más adelante, para conservar partidas con todos sus respectivos movimientos.

En cuanto a las piezas, a continuación se muestra su descripción:

- **Peón**: Es la pieza fundamental y con menor valor del juego, se disponen de 8 por jugador. Puede desplazarse sólo hacia delante y sólo de una en una casilla, pudiendo tomar piezas en diagonal, también exclusivamente en la diagonal contigua. Si un peón alcanza la última fila se debe convertir en cualquier pieza excepto en un rey o un peón. Su valor estimado es de 1.
- **Alfil**: Cada jugador dispone de dos alfiles al inicio del juego. Estos se desplazan exclusivamente en diagonal, ya sea hacia delante o hacia detrás. Su valor estimado es de 3.
- **Caballo**: Cada jugador dispone de dos caballos al inicio del juego. El movimiento del caballo es el más particular del juego. El caballo se mueve siguiendo una forma parecida a la de la letra "L". Es decir se mueve una casilla en horizontal y dos en vertical o una casilla en vertical y dos en horizontal. El caballo es la única pieza que puede desplazarse a la casilla que le permita su movimiento sin importar las piezas que haya de por medio. Su valor estimado es de 3.
- **Torre**: Cada jugador dispone de dos torres al inicio del juego. Estas se desplazan vertical y horizontalmente, ya sea hacia delante o hacia detrás. Su valor estimado es de 5.
- **Reina**: Cada jugador dispone de una reina al inicio del juego. Esta es la pieza más poderosa del juego por su inmensa movilidad aunque no la más valiosa. Puede desplazarse tanto vertical, horizontal como diagonalmente, ya sea hacia delante o hacia detrás. Su valor estimado es de 9.
- **Rey**: Es la piza más importante del juego pues si se pierde se ha perdido la partida. Cada jugador dispone de un solo rey al inicio del juego. Puede moverse tanto vertical, horizontal como diagonalmente, ya sea hacia delante o hacia

detrás, pero solo de una en una casilla. El valor estimado del rey es infinito ya que si se toma la partida finaliza.

Así quedaría la disposición del tablero de ajedrez en la posición inicial, con las piezas sobre la el tablero:

Ilustración 2 - Tablero de ajedrez en la posición inicial

Fuente: Elaboración propia mediante el editor de tableros de lichess.org

Los peones ocupan la fila 2 en el caso de las blancas y la fila 7 en el caso de las negras. El resto de las piezas por orden de aparición empezando por la casilla *a1* hasta la *a5* son: la torre, el caballo, el alfil, la reina y el rey.

El objetivo final del juego es capturar el rey del rival, o en jerga ajedrecística “dar jaque mate”.

2. Estado del arte

2.1 Orígenes del ajedrez

El juego del ajedrez se inició en India hace aproximadamente 1500 años. El nombre del juego proviene del famoso poema indio *Mahabharata*.[4] En este poema una de las formaciones que se menciona para las batallas que se luchan en él es el *chaturanga*. Este nombre significa cuatro extremidades aunque el nombre evolucionó para significar 4 tropas. Las 4 tropas eran las que estaban presentes en el juego inicial de ajedrez, éstas eran: infantería, caballería, elefantes de guerras y carros de guerra. Estas cuatro tropas son los peones, los caballos, los alfiles y las torres, a las cuales se añaden el rey y la reina (originalmente "el consejero").

Desde la India el juego fue traído a Persia dónde la palabra *chaturanga* evoluciona a *shaterej* de *shah* qué significa rey. De esta palabra deriva jaque mate, *checkmate* en inglés, *shah mat* en persa, "el rey está perdido". Una vez introducido en el mundo islámico, el ajedrez se expande por el norte de África y llega a Europa en donde toma los nombres que hoy en día conocemos como ajedrez. El juego ha tomado muchas formas hasta llegar a la forma actual pero originalmente no era muy diferente de como es ahora. Las mayores diferencias eran los movimientos de las piezas y la forma y dibujo que éstas tenían, el tablero era prácticamente igual.

El ajedrez ha fascinado a los nobles con los que se cruzaba, desde India hasta Europa. Una de las primeras menciones históricas del juego, y una de las más famosas, se da en una leyenda India. En esta leyenda, un rey indio tenía la costumbre de retar a los hombres que le visitaban a jugar al ajedrez. Un día llegó un hombre sabio y, para motivar a su oponente, el rey le ofreció una recompensa si le ganaba, esta recompensa debería ser elegida por el hombre sabio. El hombre pidió un premio muy simple: un grano de arroz por la casilla primera, dos por la segunda, cuatro por la tercera, y así sucesivamente por cada casilla del tablero.

[4] Obra de literatura india que data del S. III a. de C. de carácter bélico y mitológico.

La partida finalizó con victoria del hombre sabio y el rey mandó darle la recompensa que había solicitado. Al computar un consejero del rey la cantidad de granos que el hombre sabio había pedido, este le dijo al rey que no había tantos granos de arroz en todo su reino para poder premiar a este hombre. Este es un claro ejemplo de la relación entre el ajedrez y las matemáticas aunque sea a un nivel tan simplista. [1][2][3]

2.2 Táctica

En la sección anterior se explicó qué es el ajedrez para poder definir qué es la táctica en el ajedrez y finalmente, qué es un problema de táctica. Para poder entender qué es la táctica es conveniente conocer que la forma de jugar al ajedrez se divide en dos partes: la estrategia y la táctica.

La mejor definición de estrategia y táctica la dio Savielly Tartakower[5]: "*Táctica es saber qué hacer cuando hay algo que hacer. Estrategia es saber qué hacer cuando no hay nada que hacer*". En otras palabras, táctica es la combinación de movimientos que se deben jugar para generar una posición específica deseada, tanto de un jugador como del otro. La estrategia, por otro lado, no busca necesariamente una posición concreta, o una serie de movimientos, sin embargo busca alcanzar pequeñas ventajas basadas en los criterios que rigen la evaluación de una posición. Estos criterios serán explicados más adelante en la explicación del motor de evaluación. [4]

2.3 Problema de táctica

Una vez definidos el ajedrez y la táctica dentro de este, sólo queda responder una pregunta: ¿qué es un problema de táctica?

[5] Savielly Tartakower (Rostov del Don 1887 (Rusia) - París 1956 (Francia) Ajedrecista Polaco-Ruso, uno de los fundadores de la corriente hiper-moderna del ajedrez.

Para aquellas personas que no conozcan el juego, la respuesta más simple es *"El 'enigma' que sale en la última página de los periódicos*". Esta no es, obviamente, una definición formal. Una definición aproximada podría ser: una posición, cuyo análisis a baja profundidad muestra o paridad o ventaja a uno de los jugadores, pero en la que a través de una combinación de movimientos permite adquirir una ventaja definitiva para el otro jugador.

En esta definición *análisis a baja profundidad* significa a simple vista, al menos para los ojos de un jugador con un ELO[6] menor que el de un Gran Maestro[7]. Por otro lado se utiliza el término *ventaja definitiva*, que significa una ventaja que permita ganar la partida con comodidad.

Pero los problemas de táctica no están limitados a esta definición. Por ejemplo un problema de táctica puede darse en una posición en la que las blancas tengan una gran desventaja material y posicional, pero que mediante una combinación de movimientos consigan las tablas.

Cabe mencionar también que los problemas de táctica no tienen porqué venir de partidas reales, muchos de ellos vienen de situaciones creadas adrede para su estudio, especialmente los problemas de final de partida.

Para ilustrar al lector, a continuación se muestra un problema de táctica, en el que es el turno de las blancas:

[6] ELO es un método matemático estadístico que permite calcular la habilidad relativa de los jugadores de ajedrez.
[7] Gran Maestro es el título de mayor categoría en ajedrez. Lo adquieren aquellos jugadores que demuestran ser expertos.

Ilustración 3 - Problema 1

Fuente: lichess.org

A simple vista podría parecer que la partida está igualada, el material es igual para ambos jugadores, el rey negro tiene piezas enemigas cercanas a su posición, pero la reina blanca está clavada con lo cual la reina blanca puede darse por perdida. A pesar de que la partida pueda parecer igualada, o incluso con ventaja para las negras por la captura inminente de la reina blanca, en esta posición hay un mate en dos movimientos para las blancas. [4]

2.4 Samuel Lloyd

En cuanto al origen de los problemas de táctica y enigmas de ajedrez este es un tanto incierto pero lo que está claro es que uno de sus primeros contribuyentes a gran escala fue Samuel Lloyd.

Samuel Lloyd nació en Filadelfia el 31 de enero de 1841 aunque pronto se trasladó a Nueva York en donde atendió a la escuela. Desde pequeño mostró mucho interés por el ajedrez uniéndose a los 14 años a un club de ajedrez con sus hermanos, de hecho aún estando en la escuela ya ganó premios por los problemas que planteó de ajedrez.

Su primer problema se publicó en el *New York Saturday Courier* el 14 de abril de 1855 y el segundo que escribió le proporcionó su primer premio.

Después de este reciente éxito empezó a escribir para la revista de ajedrez *Chess Monthly.* Una vez terminada la escuela se decidió a estudiar ingeniería para convertirse en un ingeniero mecánico pero se dio cuenta de que podía ganarse la vida creando enigmas y problemas de ajedrez. Lloyd también competía al ajedrez pero no era un jugador de alta clase. Este hecho demuestra que ser bueno resolviendo problemas de ajedrez no significa ser bueno jugando al ajedrez.

Por concluir sobre Lloyd, él no solo era bueno redactando y diseñando problemas de ajedrez sino que también creaba enigmas y puzles, rompecabezas en general. Uno de los más famosos es el rompecabezas de los caballos y los jinetes.

A continuación se muestra uno de los problemas de mate de Lloyd, mueven blancas y hacen mate en 5:

Ilustración 4 - Problema 2 (S. Lloyd)

Fuente: Elaboración propia mediante el editor de tableros de lichess.org

Otros problemas que ideó Lloyd fueron los siguientes:

- Si ambos jugadores tuviesen que realizar los mismos movimientos, ¿cómo podrían blancas realizar un mate en 4 movimientos?
- Encuentra una posición de empate por ahogo en 10 movimientos
- ¿Cómo se puede dar jaque mate por ataque descubierto en 4 movimientos?

Problemas como estos, es decir poder idear y resolver estos problemas, demostraban la gran capacidad intelectual que tenía Lloyd para crear este tipo de enigmas y puzles ajedrecísticos. [5][6]

2.5 WFCC

Lloyd fue uno de los principales y primeros agentes en la composición de problemas de ajedrez, pero este campo ha evolucionado mucho desde entonces. Hoy en día existe la Federación Mundial de Composición Ajedrecística, *WFCC*[8] por sus siglas en inglés. Esta federación hace torneos anuales en los que compiten jugadores tanto para resolver problemas como para crearlos.

Diferentes aspectos entran en la valoración de quién gana y quién pierde. En lo que respecta a la resolución, un menor tiempo y un mayor número de problemas resueltos son las características principales que dan puntos. En cuanto a la composición, se valoran diversos factores como la estética, entre otros, que puntúan estos problemas.

Existen principalmente ocho tipos de composiciones de ajedrez o problemas de ajedrez que define esta federación aunque, en la realidad, hay muchos más. Estos tipos son:

- Mates en 2 movimientos
- Mates en 3 movimientos
- Mates en más de 3 movimientos
- Estudios
- Auto-mates
- Mates con ayuda
- Ajedrez de fantasía
- Retro y matemáticos

[7]

Este trabajo se centrará en las primeras tres categorías, es decir, los problemas de ajedrez en los que se da mate.

[8] WFCC: World Federation for Chess Composition

2.6 Búsqueda de problemas por ordenador

La búsqueda de problemas de táctica por ordenador va ligada a dos elementos fundamentales: la capacidad computacional de los ordenadores y la capacidad de dar evaluaciones veraces de los motores de ajedrez.

La primera aparición de una máquina jugando al ajedrez fue en el siglo XVIII. Wolfang von Kempelen[9] creó el turco mecánico y mostró esta máquina por todo el mundo. Aparentemente la máquina era simplemente una mesa grande con la maquinaria interna propiamente diseñada de forma tal que las piezas se movieren de forma lógica y automáticamente tras el movimiento del rival. Posteriormente esta máquina fue desmantelada y se reveló la verdad sobre ella. Esta era un fraude ya que en el interior de la máquina se encontraba un hombre de pequeño tamaño que poseía una gran habilidad en el juego.

Más allá de la anécdota, esta historia suscitó un enorme interés por crear una máquina que fuera capaz de jugar al ajedrez, de verdad, sin pequeños hombres debajo de la mesa.

Posteriormente, en el siglo XIX, se crearon los primeros autómatas. Máquinas capaces de realizar una determinada acción a partir de un determinado estado en función de un determinado estímulo, también llamados autómatas deterministas.

Pero realmente el ajedrez por ordenador no tuvo un verdadero reflejo hasta la llegada de Von Neumann[10] y de Turing[11]. Estas dos figuras históricas, considerados por muchos los padres de la computación moderna, dieron lugar a los primeros ordenadores.

[9] Johann Wolfgang Ritter von Kempelen de Pázmánd (Bratislava 1734 – Viena 1804) Noble y escolar húngaro, gran jugador de ajedrez que acostumbraba a jugar con la nobleza austrohúngara. Es conocido por ser el creador del Turno Mecánico.
[10] John Von Neumann (Budapest 1902 – Washington 1957), es considerado uno de los matemáticos más importantes de la era moderna por la importancia de sus trabajos en múltiples áreas.
[11] Alan Mathison Turing (Londres 1912 – Cheshire 1954), es considerado el padre de la computación e informática moderna.

Pero no fue hasta 1951 que un colega de Turing escribió el primer programa de ordenador capaz de jugar al ajedrez. Este programa sólo era capaz de resolver algunos sencillos problemas debido a las limitaciones tecnológicas del momento.

Fue también a mediados del siglo XX cuando se empezó a utilizar el algoritmo Minimax de forma más efusiva para tratar cuestiones ajedrecísticas. Este algoritmo resultó ser el mayor aliado para aquellas personas que deseaban realizar análisis de juegos de suma cero[12] como el ajedrez y especialmente para el ajedrez.

Con la mejora de las capacidades tecnológicas, tanto en memoria como en velocidad de procesamiento, surgieron los primeros programas capaces de jugar partidas completas de ajedrez. Inicialmente estos programas eran terribles en el juego, pero con el tiempo se desarrollaron programas mejores como *Deep Blue*, *Fritz*, *Stockfish*, *Komodo* o *Houdini*.

En cuanto al objetivo que trata este proyecto, la búsqueda de problemas de táctica automatizada, se trata de una aproximación relativamente reciente y sobre la que no hay prácticamente nada escrito.

La referencia más clara al respecto la da *lichess.org*. Esta página es un portal web dedicado al ajedrez. En ella se puede jugar con otros jugadores, contra el motor Stockfish, utilizar su herramienta de evaluación, etc. El aspecto de lichess que interesa a este proyecto es su sección de entrenamiento con problemas de táctica. En esta sección te muestran problemas de táctica que van cambiando de dificultad según tu habilidad. La dificultad de los problemas se determina por el porcentaje de jugadores que lo resuelven junto con su habilidad para resolver problemas (medida en ELO). Pero lo más interesante de esta herramienta es que todos los problemas que en ella se muestran provienen de partidas jugadas en la propia página. Es decir esta página realiza con éxito el propósito mismo de este proyecto, y aparentemente a gran escala.

[12] Los juegos de suma cero son aquellos en los que la ganancia o pérdida por parte de un jugador es contrarrestada por la ganancia o perdida de los demás jugadores.

Otra página interesante en este aspecto es *chesstempo.com*. Esta página es muy similar a la anterior, pero su herramienta de problemas de táctica es aún más selecta, pudiendo determinar el tipo de partidas de las que se extraen los problemas. [8][9][10][11][12]

3. Algoritmos de búsqueda en el ajedrez

3.1 Algoritmo Minimax

Minimax es uno de los algoritmos de búsqueda más utilizados en el ajedrez. Es el algoritmo que utiliza el motor de ajedrez Stockfish[13]. A continuación se explicará su funcionamiento.

3.1.1 Definición

Es un algoritmo recursivo que se ideó originalmente para juegos de dos personas. En este algoritmo se trata de maximizar el número de veces que gana uno de los jugadores, o en otras palabras, maximizar los puntos que tiene uno de los jugadores con respecto al contrincante. Para ello, analiza movimiento a movimiento maximizando los puntos de un jugador en su turno y en el turno del rival minimizando los puntos es decir maximizando los puntos del rival.

El algoritmo Minimax, realiza para cada nodo el siguiente cálculo:

- Si es un nodo **terminal**: Calcula el valor con la función de evaluación.
- Si es un nodo **Max**: Toma el máximo valor de todas las acciones posibles.
- Si es un nodo **Min**: Toma el mínimo valor de todas las acciones posibles.

En la lista anterior aparece la expresión función de evaluación. Esta función no es más que la forma en la que se calculan los puntos de la partida. Por ejemplo en el juego de *las tres en raya* se podría crear una función de evaluación que devuelva 1 si las *X* ganan, -1 si ganan las *O* y 0 en caso de empate.

El algoritmo comienza evaluando los nodos terminales o nodos hoja con la función de evaluación y arrastra la evaluación o puntuación de esos nodos hacia arriba en el árbol siguiendo las reglas listadas anteriormente.

[13] Stockfish es un motor de ajedrez UCI (Interfaz de ajedrez Universal) de código abierto.

3.1.2 Ejemplo

Para comprender mejor este algoritmo a continuación se muestra un ejemplo en el que se tomará la siguiente posición, se busca calcular la evaluación para esta posición a profundidad 2 (mueven blancas).

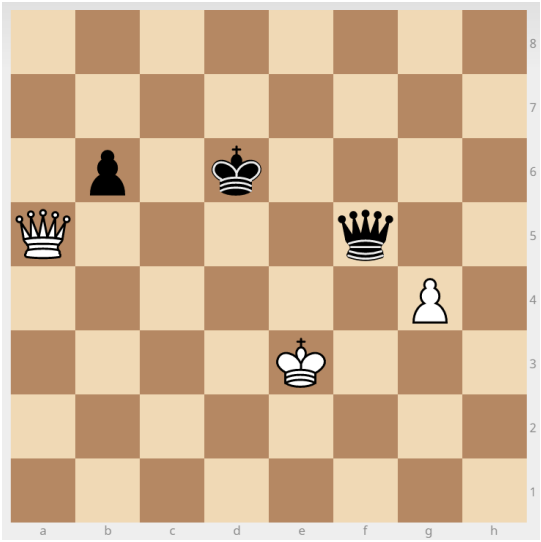

Ilustración 5 - Posición del nodo Max, nodo raíz

Fuente: Elaboración propia mediante el editor de tableros de lichess.org

En el algoritmo Minimax, los nodos que se encuentran a profundidad 0 o par son los nodos Max, que en este caso representan a las blancas (turno de mover para las blancas). Por otro lado, los nodos que se encuentran en profundidad impar son los nodos Mini, que en este caso representan a las negras (turno de mover para las negras).

Para realizar este ejemplo no se considerarán todos los movimientos posibles ya que la anchura del árbol sería demasiado grande. Se considerarán exclusivamente tres movimientos en el nodo raíz (la posición representada en la figura superior) y dos movimientos en los subsiguientes nodos. De esta forma se obtienen seis nodos terminales o nodos hoja.

El diagrama del algoritmo Minimax, en su estado inicial quedaría representado de la siguiente forma:

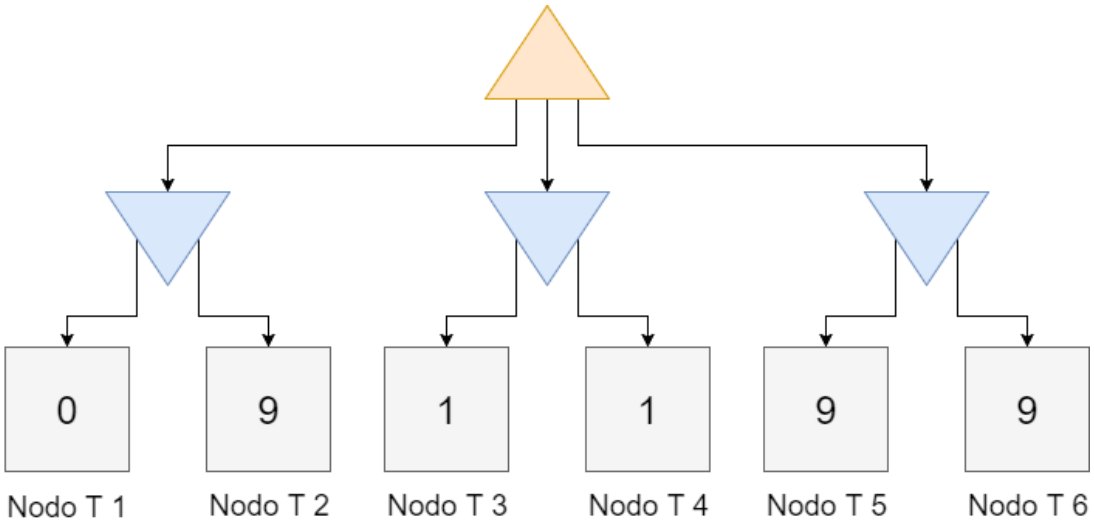

Ilustración 6 - Árbol en el paso 0

Fuente: Elaboración propia mediante el editor de tableros de lichess.org

Los números en los nodos hoja (nombrados *Nodo T n*) son los valores de la fórmula de evaluación para cada nodo (posición). Estos nodos son también nodos Max. En este caso, para simplificar el problema, la fórmula de evaluación es simplemente el valor de las piezas blancas en el tablero menos el valor de las piezas negras en el tablero (ver el punto 1 para el valor de las piezas):

$$f(posición) = ValorPiezas(blancas) - ValorPiezas(negras)$$

De acuerdo a este diagrama, el nodo Max, que busca maximizar la puntuación, corresponde a la posición inicial, provista anteriormente. El primer nodo Min, el

primero empezando por la izquierda corresponde a la posición tras el movimiento *g4xf5*:

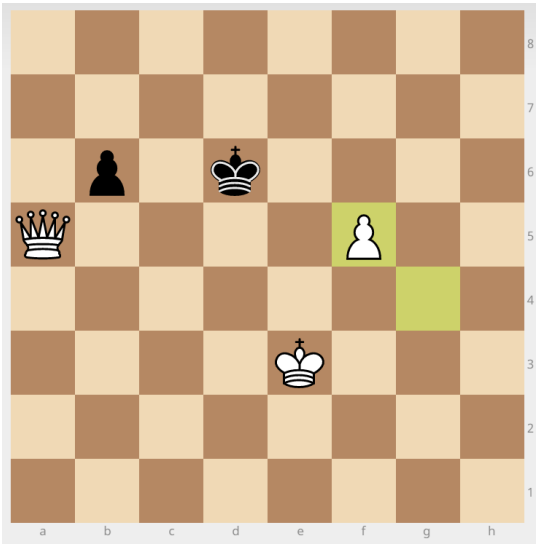

Ilustración 7 - Posición tras *g4xf5*

Fuente: Elaboración propia mediante el editor de tableros de lichess.org

El segundo nodo Min, (segundo empezando por la izquierda) corresponde a la posición tras el movimiento *Qxb6*:

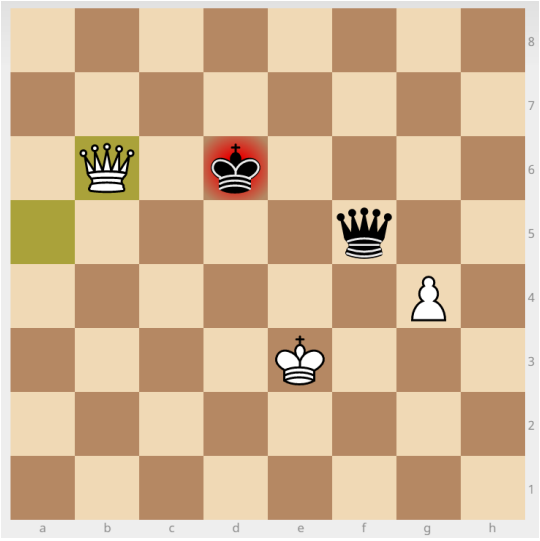

Ilustración 8 - Posición tras *Qxb6*

Fuente: Elaboración propia mediante el editor de tableros de lichess.org

El tercer nodo Min, (primero empezando por la derecha) corresponde a la posición tras el movimiento *Qxf5*:

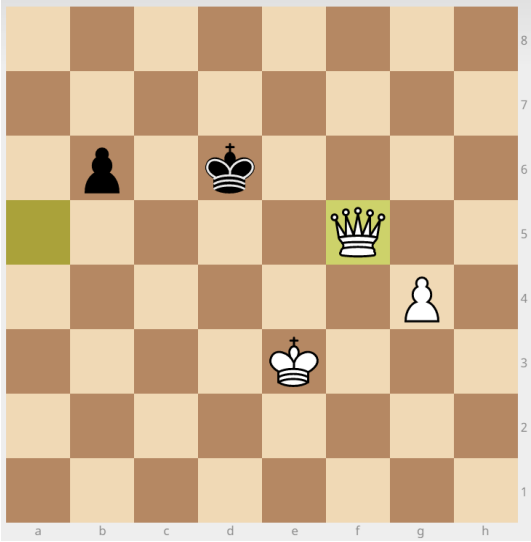

Ilustración 9 - Posición tras *Qxf5*

Fuente: Elaboración propia mediante el editor de tableros de lichess.org

Es decir que en sólo consideramos los movimientos *g4xf5*, *Qxb6* y *Qxf5* de entre los 21 movimientos posibles. Si empleásemos el algoritmo íntegramente se deberían estudiar las 21 ramas del árbol. Por último como se mencionó previamente, para los nodos Min sólo se considerarán dos movimientos. Para el primer nodo Min se consideran los movimientos *bxa5* y *Kc6*. Las posiciones resultantes respectivamente serían:

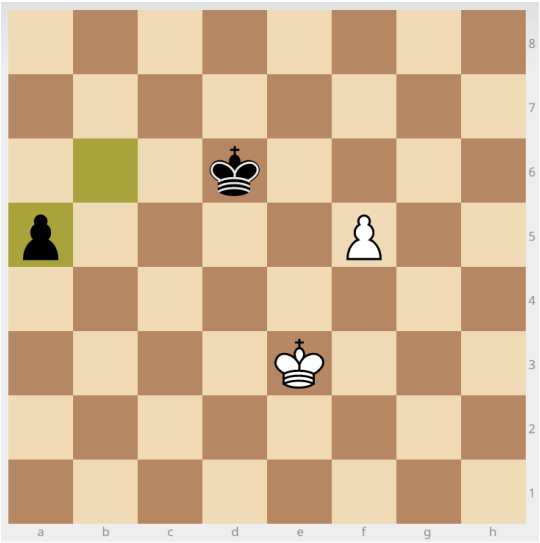

Ilustración 10 - Posición tras *bxa5*

Fuente: Elaboración propia mediante el editor de tableros de lichess.org

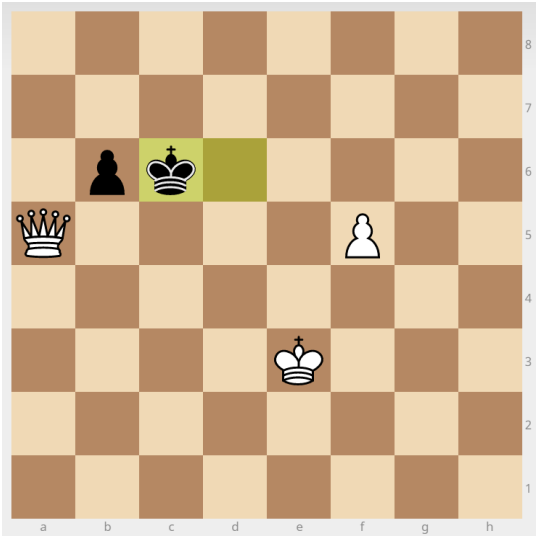

Ilustración 11 - Posición tras *Kc6*

Fuente: Elaboración propia mediante el editor de tableros de lichess.org

Estas dos posiciones corresponden con los nodos hoja *Nodo T 1* y *Nodo T 2*. La puntuación del *Nodo T 1* es 0, ya que cada peón vale un punto y ambos jugadores tienen uno. Sin embargo para el *Nodo T 2* el valor es 9, porque ambos tienen un peón

pero las blancas tienen una reina. Así quedaría la fórmula para cada una de estas posiciones:

$$f(Nodo\ T\ 1) = (Peón) - (Peón)$$

$$f(Nodo\ T\ 1) = (1) - (1) = 0$$

$$f(Nodo\ T\ 2) = (Peón + Reina) - (Peón)$$

$$f(Nodo\ T\ 2) = (1 + 9) - (1) = 9$$

Una vez calculados los valores de todos los nodos terminales se procede a calcular el valor del primer nodo Min, si la profundidad fuese mayor se calcularía el valor del primer nodo a profundidad *p-1* siendo *p* la profundidad a la que se realiza la búsqueda.

Como este nodo es un nodo Min, este tomará el mínimo valor de las acciones posibles, en este caso el nodo tras la acción *bxa5* tiene valor 0, y el nodo tras la acción *Kc6* tiene valor 9, por lo tanto el nodo Min toma el valor más pequeño, 0.

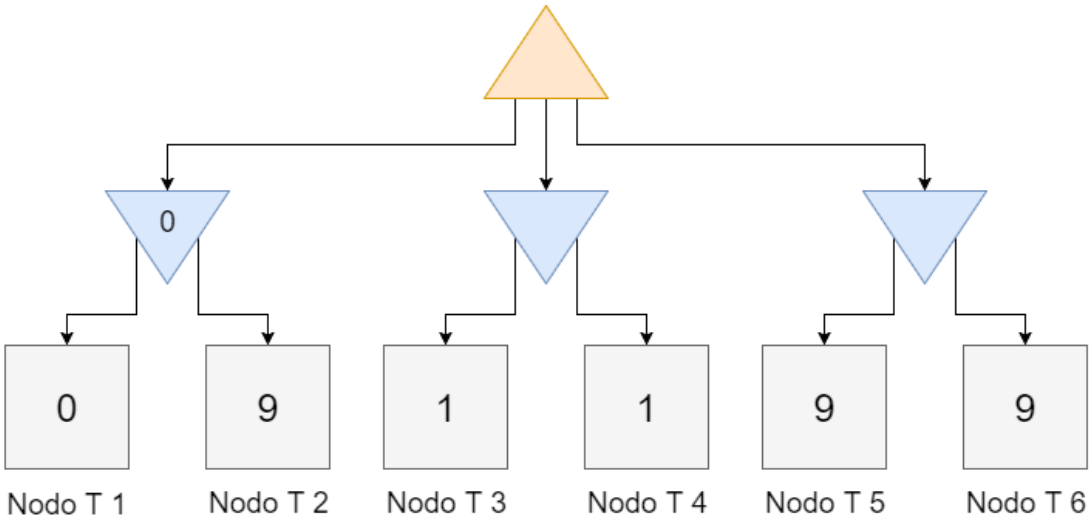

Ilustración 12 - Árbol en el paso 1

Fuente: Elaboración propia

Esta tarea de seleccionar la puntuación menor en los nodos Min significa que el algoritmo considera el mejor movimiento del jugador Min (negras). Este valor obtenido se pasaría al nodo Max superior, ya que no hay ningún otro nodo con el que comparar (todavía).

En el siguiente nodo Min las acciones que se consideran son las siguientes *Ke5* y *Kd7*, estas dejan las posiciones de los nodos *Nodo T 3* y *Nodo T 4*:

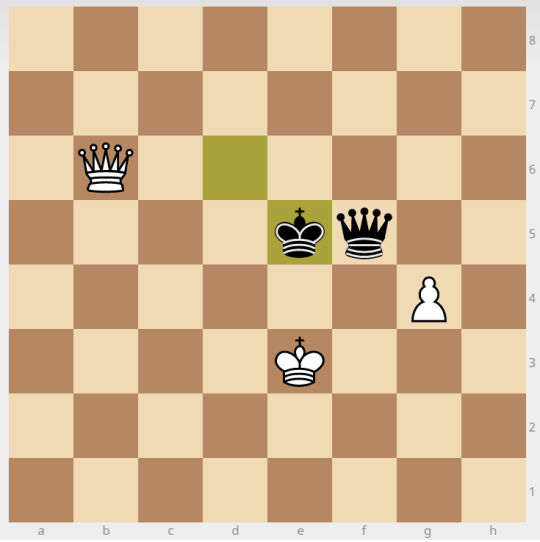

Ilustración 13 - Posición tras *Ke5*

Fuente: Elaboración propia mediante el editor de tableros de lichess.org

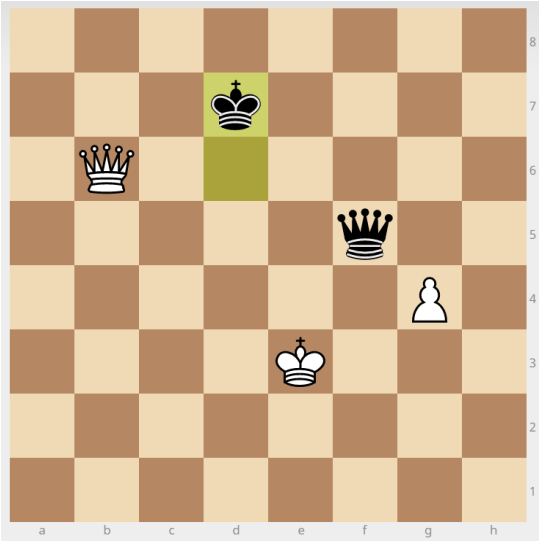

Ilustración 14 - Posición tras *Kd7*

Fuente: Elaboración propia mediante el editor de tableros de lichess.org

El valor de estos dos nodos es 1, valor calculado con la función de evaluación en el paso anterior ya que son nodos terminales. Por lo tanto el segundo nodo Min tomará el mayor de entre estos dos valores, que en este caso es 1.

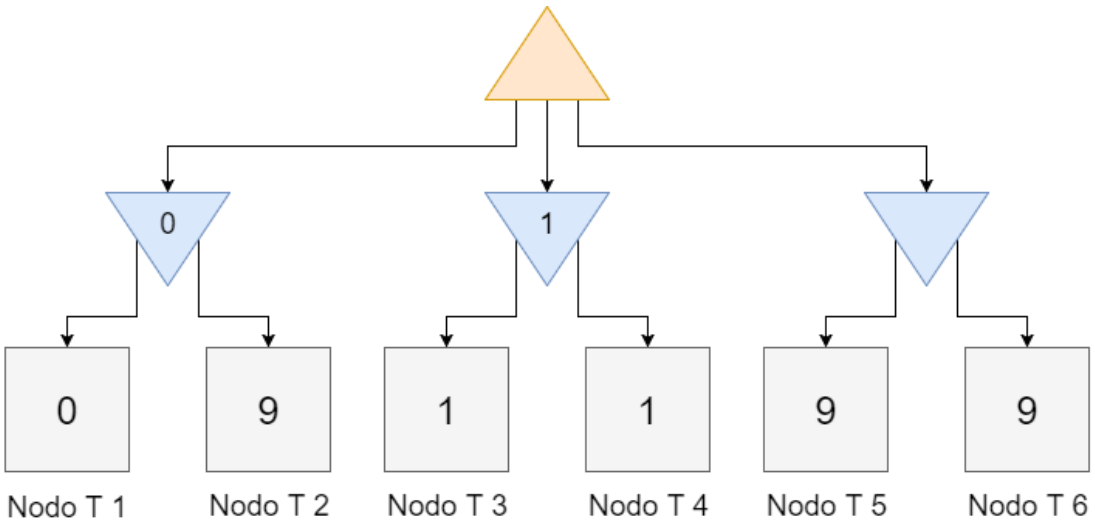

Ilustración 15 - Árbol en el paso 2

Fuente: Elaboración propia

Tras esta acción el nodo Max superior pasaría a tomar el valor 1, ya que tenía previamente el valor 0 y 1 > 0.

Queda el último nodo Min por calcular, en este caso las posiciones tras las acciones que se consideran para este nodo, *Kc6* y *Ke7* son las siguientes:

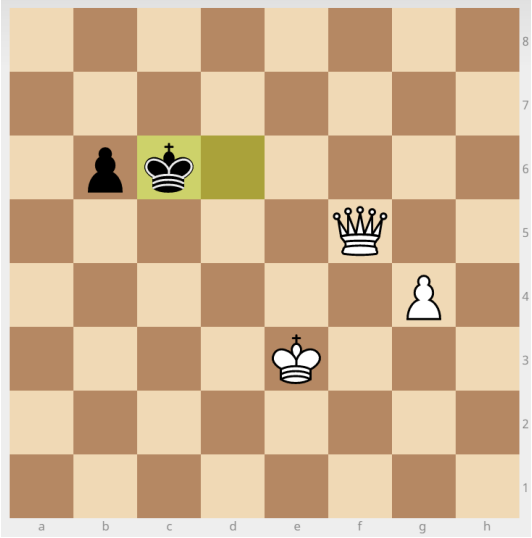

Ilustración 16 - Posición tras *Kc6*

Fuente: Elaboración propia mediante el editor de tableros de lichess.org

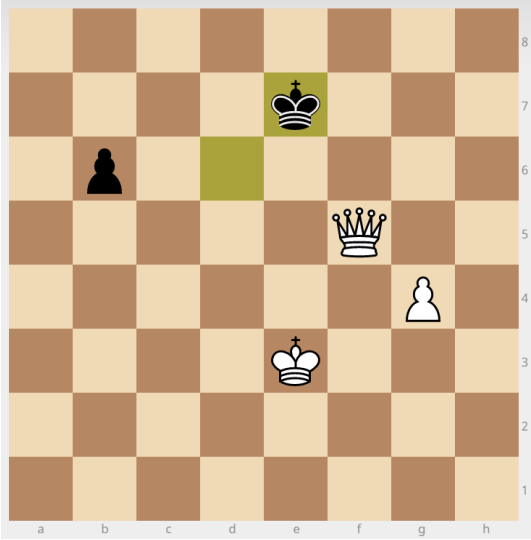

Ilustración 17 - Posición tras *Ke7*

Fuente: Elaboración propia mediante el editor de tableros de lichess.org

Para ambas posiciones la puntuación es 9, calculada con la función de evaluación. Por lo tanto el nodo Min tomará el mayor de entre estas dos, que en este caso es 9.

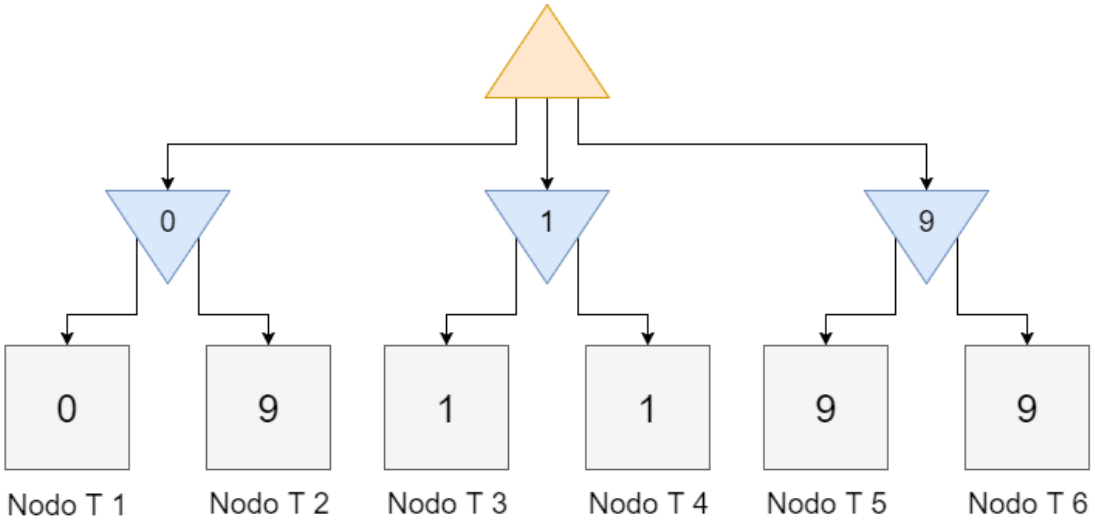

Ilustración 18 - Árbol en el paso 3

Fuente: Elaboración propia

Una vez se han calculado todos los nodos Min se pasa a calcular el nodo Max. Este nodo tomará el mayor valor de todas las acciones que puede tomar. Para este ejemplo se han considerado:

- *g4xf5,* primer nodo Min, valor 0
- *Qxb6*, segundo nodo Min, valor 1
- *Qxf5*, tercer nodo Min, valor 9

El mayor de estos tres nodos es 9 y por lo tanto este será el valor que tome el nodo Max, y así quedaría el árbol completo:

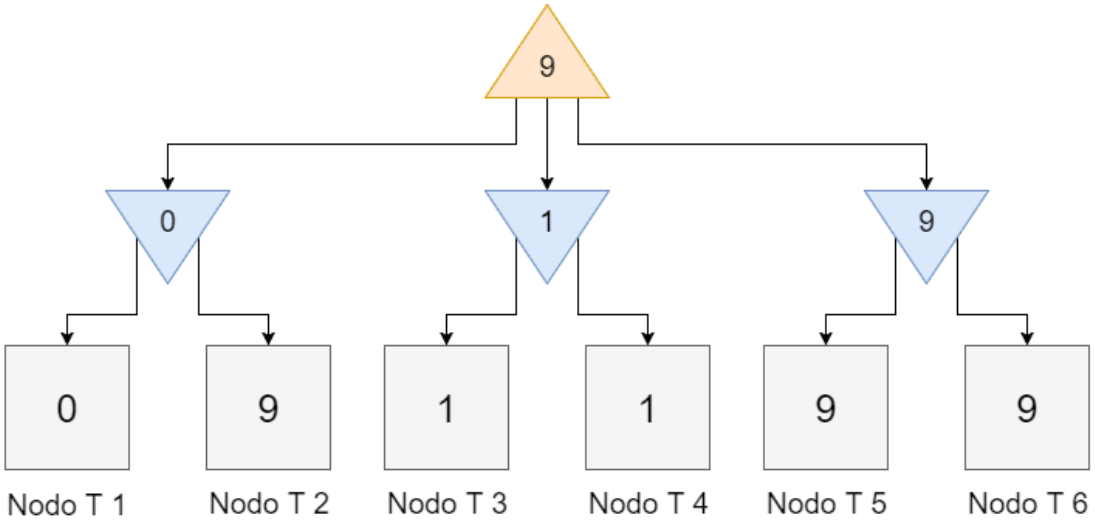

Ilustración 19 - Árbol en el paso 4

Fuente: Elaboración propia

Por si el ejemplo no ha quedado claro a continuación se muestra el árbol con cada acción representada en cada rama y cada posición en cada nodo:

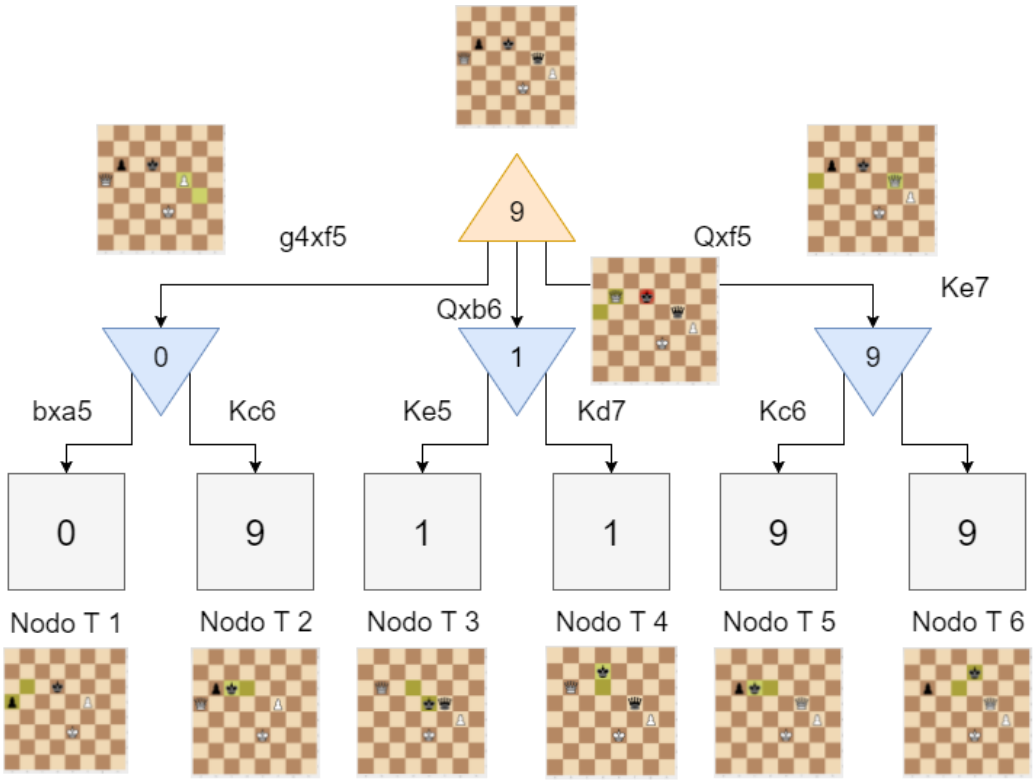

Ilustración 20 - Árbol final detallado

Fuente: Elaboración propia junto con el editor de tableros de lichess.org

La complejidad computacional de este algoritmo es exponencial. Con el objetivo de mejorar la eficiencia del mismo nace la poda alfa-beta que se explica a continuación. [13][14][15]

3.2 Poda Alfa-Beta

3.2.1 Definición

El objetivo del algoritmo alfa-beta es conseguir el valor correcto de la función Minimax sin tener que explorar todos los nodos del árbol. Esto se consigue mediante la poda de determinadas ramas que se consideran inútiles para la resolución del algoritmo porque no alterarían el valor de los nodos superiores.

La poda alfa-beta se basa en los valores Alfa y Beta. El valor Alfa es el valor de la mejor acción que se ha encontrado en cualquier punto de la búsqueda para Max hasta ese nodo. Por otro lado, el valor Beta es el valor de la mejor acción que se ha encontrado en cualquier punto de la búsqueda para Min hasta ese nodo. Es decir hay un par de valores Alfa-Beta para cada nodo, exceptuando los nodos hoja.

El algoritmo, para cada nodo explorado, comprobará si puede mejorar los valores de alfa o beta, de ser así los cambia. Si en una rama se da un valor que impide la mejoría del nodo superior, esa rama se poda y por lo tanto se deja de explorar. Esto es, si tienes un valor de alfa x en un nodo Max a profundidad p y explorando una de las ramas, en un nodo Max a profundidad $\mathrm{p}+2$, encuentras un valor y tal que $\mathrm{y}<x$, esa rama se podaría. Esto se debe a que independientemente del resto de nodos de la rama, el nodo Min que se encuentra a profundidad $\mathrm{p}+1$, del cual parte la rama que se está explorando, nunca va a tomar un valor mayor de x porque ya ha encontrado uno menor (y).

Esto significa que el orden en el que se exploren los nodos influirá en gran manera a este algoritmo. Ya que si se exploran primero los nodos que por su valor podan las ramas la eficiencia mejorará, sin embargo si estos son los últimos en explorarse no habrá mejora alguna.

3.2.2 Ejemplo

Como se realizó con el algoritmo Minimax a continuación se muestra un ejemplo de la aplicación del algoritmo con la poda Alfa-Beta.

El árbol será idéntico al del ejemplo anterior pero en este caso se explorará el árbol de derecha a izquierda. Por lo tanto primero se explora la rama que contiene los nodos Nodo T 5 y Nodo T 6.

Todos los nodos originalmente tienen los valores alfa a infinito negativo y beta a infinito positivo, excepto los nodos hoja que no tienen valores alfa-beta. Por lo tanto se empieza a explorar la rama derecha:

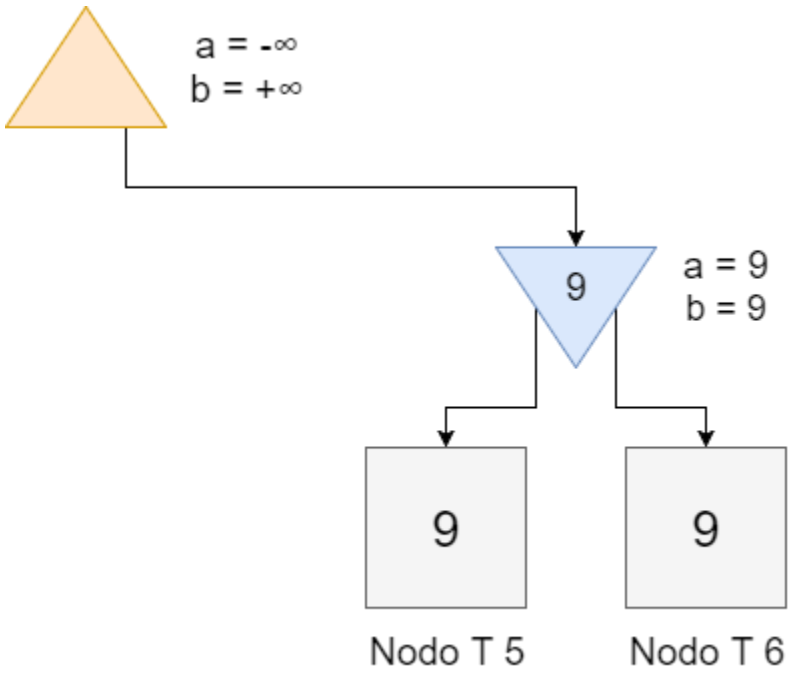

Ilustración 21 - Poda paso 1

Fuente: Elaboración propia

El nodo Min tomaría los valores alfa igual a 9 y beta igual a 9. Ya que tanto el máximo como el mínimo valor de las acciones exploradas hasta ese nodo tienen una evaluación de 9. El valor del nodo se determina con el algoritmo de Minimax, y es 9.

A continuación se pasaría el valor de alfa al nodo Max superior para poder podar ramas futuras:

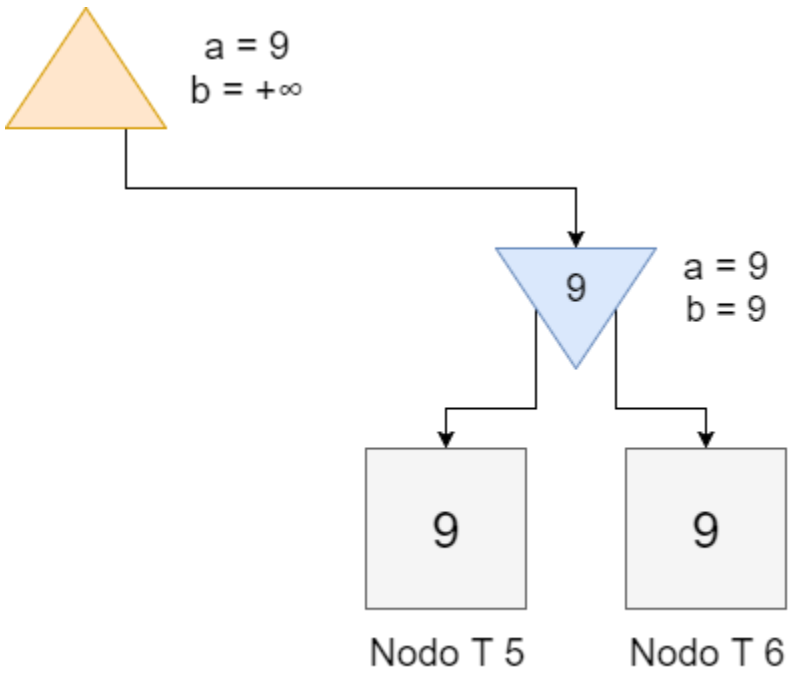

Ilustración 22 - Poda paso 2

Fuente: Elaboración propia

Se procede a explorar la siguiente rama, la rama central. Primero el nodo de la derecha, *Nodo T 4*. El nodo Min hereda un valor de alfa igual a 9 y de beta infinito negativo. El nodo pasaría un valor de beta de 1 por lo tanto el valor que el nodo Min

cogería como máximo será 1, valor que no mejora alfa (9). Por esta razón esta rama se podaría y se dejaría de explorar.

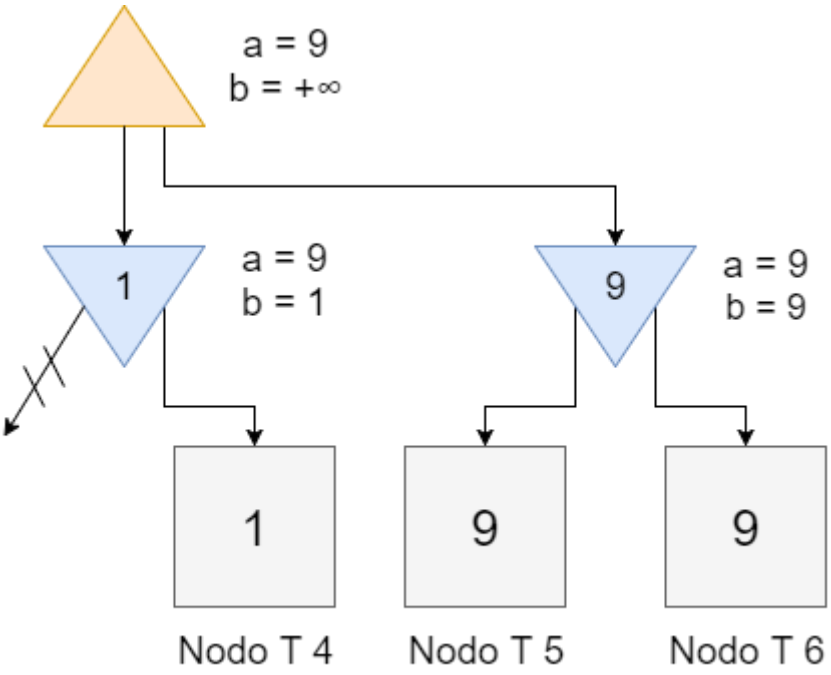

Ilustración 23 - Poda paso 3

Fuente: Elaboración propia

Finalmente se exploraría la última rama empezando por la derecha. El nodo Min en esta rama hereda un valor de alfa de 9 y de beta infinito negativo. Esta exploraría el nodo Nodo T 2 encontrando el valor 9 y pasándolo a beta. Como este nodo ya no puede mejorar alfa (9) ya que como máximo el valor del nodo será 9, la rama se podaría y se dejaría de explorar.

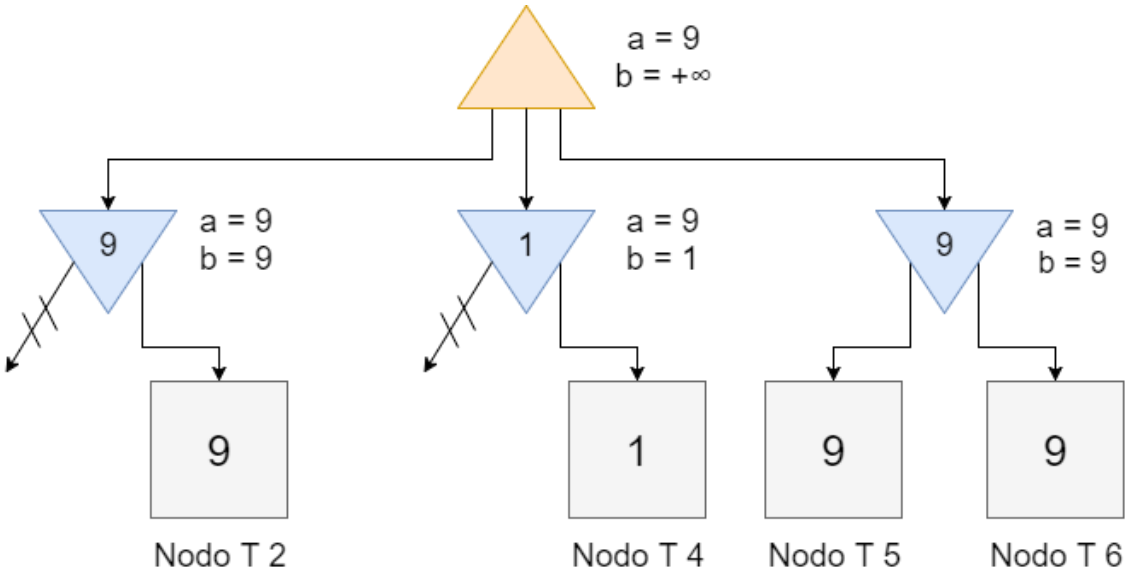

Ilustración 24 - Poda paso 4

Fuente: Elaboración propia

Por último estos valores se pasarían al nodo Max raíz quedando el árbol:

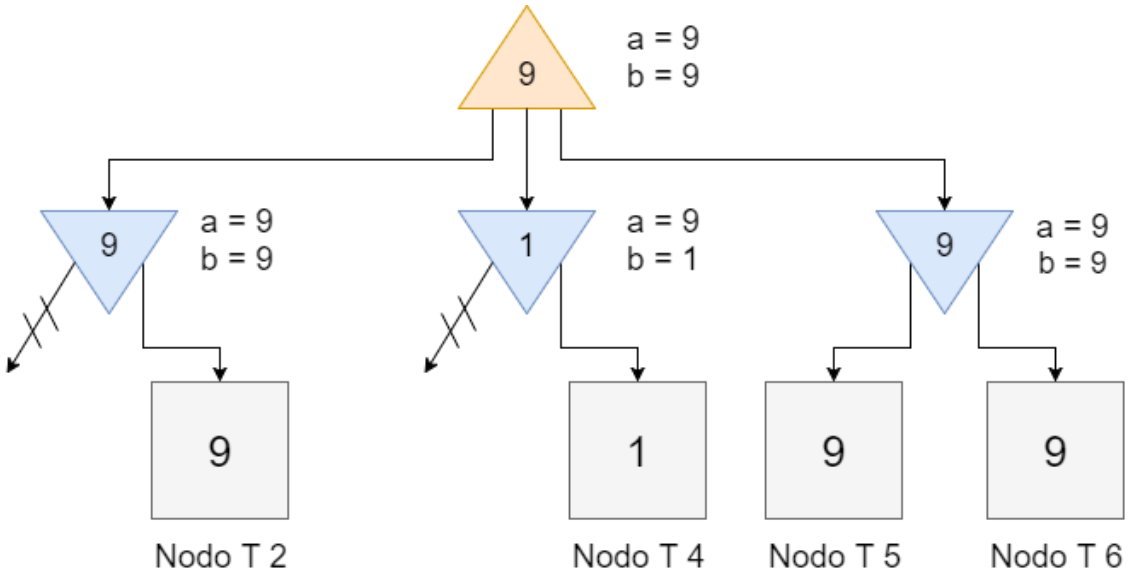

Ilustración 25 - Poda, paso final

Fuente: Elaboración propia

Como se puede comprobar este algoritmo ha encontrado la misma solución que el Minimax simple pero explorando dos nodos menos. [16][17]

4. Solución

Antes de explicar el código al detalle es necesario conocer dos herramientas que se utilizan en el mismo: Stockfish y Pgn-Extract.

4.1 Stockfish

Stockfish es el motor de análisis utilizado para la evaluación de cada posición. Este es un motor de distribución libre, empleado y diseñado para el ajedrez. Actualmente tiene un rating ELO de 3400 (el ELO de Magnus Carlsen[14] es de 2827[15]), lo que lo convierte en el mejor "jugador" del mundo.

Para poder utilizar Stockfish se adapta y usa el protocolo UCI. El protocolo UCI o Interfaz Universal de Ajedrez (por sus siglas en inglés) es un protocolo que siguen muchos motores de ajedrez para tratar de conseguir cierta universalidad, de forma que sean más sencillos de utilizar y de probar, especialmente unos contra otros.

El algoritmo que utiliza Stockfish es el algoritmo Minimax con la poda Alfa-Beta. Pero lo más interesante de este motor es la función de evaluación que utiliza.

Esta función de evaluación tiene diferentes capas. La primera capa, la superior, evalúa la posición como fase intermedia y como fase final. Con los dos valores que devuelven estas dos evaluaciones realiza una suma ponderada, en la que la ponderación depende de la fase de juego. Por ejemplo si sólo quedan dos peones y el rey para cada jugador, la evaluación de fase final cobrará mayor importancia mientras que la de fase intermedia apenas tendrá relevancia.

La siguiente capa corresponde a la evaluación específica que realiza de la fase intermedia y de la fase final. En la evaluación de estas dos fases se consideran los siguientes factores:

[14] Magnus Carlsen (Akershus 1990 , Noruega) Ajedrecista noruego, segundo campeón mundial más joven de la historia. Es el actual campeón del mundo y se sitúa en el puesto número 1 en el ranking por ELO con 2826 puntos.
[15] https://ratings.fide.com a 15 de septiembre de 2017.

- Valor de las piezas: Este valor es parecido al definido en el punto 1 aunque a otra escala (un peón vale 181).
- Desarrollo de las piezas: Si una pieza ocupa la casilla original la evaluación será menor que si está ya ha sido desarrollada.
- Par de alfiles: Si el jugador tiene la pareja de alfiles obtendrá una evaluación mayor que si tiene un alfil y un caballo.
- Estructura de los peones: Los peones aislados, doblados y peones que no están defendidos por otro peón obtienen una evaluación menor que los peones conectados. Los peones pasados obtienen una evaluación mayor y si estos son dos o más y conectados aún mayor.
- Posición de las piezas:
    - Las piezas menores protegidas por un peón obtienen una evaluación mayor.
    - Los caballos que se encuentran protegidos por un peón en una posición en la que no pueden ser atacados por ningún peón enemigo (outpost[16]) obtienen una evaluación mayor. El resto de piezas también obtienen mayor evaluación si se da este caso, pero en menor medida (el orden de bonus es caballo, alfil, torre y reina)
    - Las torres que no se puedan mover obtienen una evaluación menor.
    - Las torres en columnas abiertas[17] o columnas semi-abiertas[18] obtienen una evaluación mayor.
    - Los alfiles que estén en las diagonales mayores obtienen una evaluación mayor.
    - Los peones cercanos a las casillas de coronación, especialmente si no tienen oposición, obtienen una mayor evaluación.
- Movilidad de las piezas: Cuanto mayor número de movimientos pueda realizar una pieza, mayor será su evaluación.

[16] Outpost: En ajedrez se da cuando existe una pieza menor protegida por un peón en una posición en la que la pieza menor no puede ser atacada por un peón.
[17] Columna abierta: en ajedrez es aquella columna que no tiene peones.
[18] Columna semi-abierta: en ajedrez es aquella columna que solo tiene un peón.

- Amenazas: Las piezas que no estén defendidas y bajo la amenaza de una pieza enemiga reciben una evaluación menor.
- Protección del rey: Cuanto mayor sea la amenaza al rey de un jugador, menor será su evaluación. La amenaza incluye piezas atacando casillas cercanas al rey, piezas aliadas que no estén protegiendo al rey, peones cerca del rey, etc.

Dependiendo de si se trata de la evaluación de final de partida o de fase intermedia, los diferentes factores tendrán más o menos ponderación.

En la lista anterior no se muestran todos los factores que incluye la función de evaluación pero sí los más importantes. Cualquier persona que haya sido instruida en el ajedrez sabrá que los conceptos que evalúa Stockfish no son revolucionarios. Estos conceptos son, en su gran mayoría, los mismos conceptos que se enseñan en las escuelas de ajedrez. Controla el centro, no aísles peones, mantén el par de alfiles, son frases que los alumnos de ajedrez escuchan repetidas veces durante su aprendizaje y son los valores estratégicos fundamentales del ajedrez. [18][19][20]

4.2 Pgn-Extract

El programa pgn-extract es un programa desarrollado por David J. Barnes de la Universidad de Kent. Este programa sirve para traducir partidas de ajedrez de un formato a otro. Estos formatos pueden ser FEN[19], PGN[20] o UCI[21] principalmente. En este proyecto se utiliza para traducir partidas de formato PGN a formato UCI. [21]

4.3 Lógica

El código del proyecto se divide principalmente en tres partes. La primera parte consiste en transformar las partidas del formato original que suele ser PGN a un formato comprensible por el motor Stockfish. En este caso el motor Stockfish utilizará el protocolo de ajedrez UCI.

[19] FEN: acrónimo de Forsyth-Edwards Notation (Notación Forsyth-Edwards)
[20] PGN: acrónimo de Portable Game Notation (Notación portátil de partida)
[21] UCI: acrónimo de Universal Chess Interface (Interfaz Universal de Ajedrez)

El protocolo de ajedrez UCI, interfaz universal de ajedrez por sus siglas en inglés, es un protocolo que utilizan todos los motores de ajedrez reconocidos. Se creó principalmente con dos motivaciones: la primera para poder realizar competiciones entre motores de ajedrez y la segunda motivación para facilitar el uso y acceso a los motores de ajedrez a todo tipo de usuarios. El protocolo de ajedrez UCI comprende dos formatos, el formato FEN y un formato conocido también como formato UCI que es igual que el formato PGN pero incluyendo exclusivamente la casilla de salida y la casilla de llegada de cada movimiento. [22]

Una vez transformadas las partidas del formato original a formato UCI estas se pasan a la segunda parte del programa: la evaluación. Es en esta parte en la que se pasará movimiento a movimiento y partida a partida cada posición para que sea analizada por el motor Stockfish. Esto se hará partiendo de la posición inicial e incluyendo uno a uno todos los movimientos de cada partida, otra posible solución para que Stockfish analice las partidas sería pasarle cada posición de cada partida en formato FEN, se utilizará la primera por simplificar.

Una vez se ha realizado una evaluación de todas las posiciones de todas las partidas se guarda la salida que te devuelve el motor, es decir la puntuación de todos las posiciones, y se pasa a analizar los resultados. Antes de describir la fase de análisis de resultados conviene comprender los parámetros de la evaluación. El parámetro principal de la valuación es la profundidad a la que se estudia cada posición. Tras diversas pruebas la mejor profundidad encontrada para los objetivos de este proyecto es 7, en la parte de análisis y en la sección de pruebas se verá la razón detrás de esta decisión.

Con la evaluación ya realizada de cada posición se procede a realizar el análisis. El análisis conlleva una serie de reglas de expresiones regulares en las que se extrae la puntuación a profundidad 1 y a profundidad 7 de cada posición. Con estas dos puntuaciones se computa la diferencia entre una y la otra para comprobar si existe un problema de táctica.

Lo que hace realmente el programa desarrollado es comprobar si a profundidad 1 hay una ventaja menor a 5 puntos y a profundidad 7 existe un mate. Como se mencionó anteriormente existen muchos tipos de problemas de táctica pero con esta búsqueda encontraremos sólo aquellos problemas que sean de mate en 4 o menos.

Durante la realización de este proyecto se probaron diferentes configuraciones para la fórmula que determina si en una posición se da un problema de táctica o no.

Por ejemplo, se probó calculando exclusivamente la diferencia entre la puntuación a una profundidad 5 y una profundidad 3. Realizando este análisis se devuelven gran cantidad de posiciones, algunas de ellas se podrían categorizar como problemas de táctica, aunque no de mate, pero muchas otras no. Como se menciona anteriormente este proyecto se limita a encontrar los problemas de táctica en los que se dé mate en 4 o menos movimientos, por la única razón de que estos tienen mayor certeza de ser categorizados como problemas de táctica.

4.4 Código

Todo el código está desarrollado en el lenguaje de programación Python excepto por una parte del código escrito en el lenguaje bash. El código se desarrolló siguiendo las directrices de la programación modular, es decir, separando cada función del código en una función diferente dentro de éste, incluso se ha dividido el código en tres archivos diferentes, uno para cada parte del código más el archivo con el código en bash.

El sistema de archivos quedaría de la siguiente forma:

- problemFinder.py
    - evaluate.py
        - script.sh
    - analizer.py

Es decir, el archivo *problemFinder.py* importa las funciones de *evaluate.py* y *analizer.py*. A su vez el archivo *evaluate.py* ejecuta el archivo *script.sh*.

Este último archivo (el archivo bash) se encarga de las llamadas a la evaluación por parte de Stockfish. En este código se pueden observar los diferentes comandos que toma el motor para realizar la evaluación:

```
#!/bin/bash
../Stockfish/src/stockfish << EOF
uci
ucinewgame
position startpos moves $1
go depth 7
go nodes 1
quit
EOF
```

Los dos primeros comandos tras la declaración del intérprete y la llamada a stockfish (*uci* y *ucinewgame*) sirven para iniciar el motor en formato UCI y para iniciar una nueva partida. En el tercer comando (*position startpos moves $1*) se le pasa al motor la posición que se desea analizar. Esto se hace pasándole la posición inicial y diciéndole a partir de esa posición qué movimientos debe de ejecutar. Por último, se le dice que explore hasta profundidad 7 (*go depth 7*). La última línea antes de terminar el Here-Document (*go nodes 1*) se utiliza para que no finalice la ejecución antes de finalizar el análisis a profundidad 7.

El archivo *problemFinder.py* es el archivo encargado de realizar las llamadas a las diferentes funciones y manejar la salida y entrada de las mismas:

```python
import sys
import time
from evaluate import transform_into_uci, reformat, arraify, evaluate_sf
from analizer import analize

def main():

    start = time.time()

    gamesInPgn = transform_into_uci(sys.argv[1])

    listOfGames = reformat(gamesInPgn)

    gamesAndMoves = arraify(listOfGames)

    output = evaluate_sf(gamesAndMoves)

    analize(output)

    finish = time.time()

    print "TIME ELAPSED: %s s" % (round(finish-start))

if __name__ == "__main__":
    main()
```

Código 1 - problemFinder.py

Este archivo también incluye el cálculo del tiempo de ejecución del programa. Este archivo realiza las siguientes funciones:

1. Llama a la función de traducción al formato UCI.
2. Reformatea las partidas para convertirlas en una lista de partidas
3. Convierte esa lista de partidas en una matriz donde cada fila es una partida y cada columna un movimiento.
4. Evalúa cada movimiento de cada partida.
5. Analiza los resultados para mostrar por pantalla las posiciones que incluyen un problema de táctica.

El archivo *evaluate.py* se encarga de los primeros 4 puntos, la primera función se define a continuación:

```
def transform_into_uci(file_with_games):

    arg_for_bash = "-f" + file_with_games
    p = subprocess.Popen(["pgn-extract", arg_for_bash,"-Wuci","-C"], stdout=subprocess.PIPE)
    out, err = p.communicate()

    return out
```

Código 2 - evaluate.py, transform_into_uci

Esta función es la encargada de la llamada a *pgn-extract* para transformar las partidas a formato UCI. No es necesario realizar varias llamadas porque el programa es capaz de procesar múltiples partidas.

Como se puede observar en *problemFinder.py* el argumento que recibe esta función es un documento que incluye el nombre de cada una de las partidas que se desea analizar.

Este documento puede contener por ejemplo:

Kasparov-Carlsen.pgn
Anand-Karjiakin.pgn
game3-Linares2010-Finals.pgn

En este caso transformaría las partidas con esos tres nombres si existen en el directorio.

La segunda función quedaría definida a continuación:

```
def reformat(gamesInPgn):

    a = gamesInPgn
    b = re.sub('\[.*\]','',a)
    c = re.sub('\n+','\n',b)
    d = re.sub('1-0\n|0-1\n|1/2-1/2\n','GAME',c)

    listOfGames = d.split('GAME')
    listOfGames.pop()

    return listOfGames
```

Código 3 - evaluate.py, reformat

Esta función se encarga mediante expresiones regulares de transformar la salida de *pgn-extract* en una lista de *python*.

La tercera función se define como sigue:

```
def arraify(listOfGames):

    gamesAndMoves = []
    for game in listOfGames:
        a = game.split(' ')
        a.pop()
        gamesAndMoves.append(a)
    gamesAndMoves[0][0] = gamesAndMoves[0][0].split("\n")[1]

    return gamesAndMoves
```

Código 4 - evaluate.py, arraify

Esta función a su vez transforma la lista obtenida en la función anterior en una matriz que, como se mencionó anteriormente, contendrá una partida por cada fila y un movimiento por cada columna en cada fila.

Por último en el archivo evaluate.py queda la función *evaluate_sf*:

```
def evaluate_sf(gamesAndMoves):

    gameCounter = 0
    output = ""

    for game in gamesAndMoves:
        gameCounter = gameCounter + 1
        output = output + "GAME" + str(gameCounter)
        movex = ""

        for move in game:
            movex = movex + " " + move
            p = subprocess.Popen(["../Stockfish/src/script.sh", movex], stdout=subprocess.PIPE)
            out, err = p.communicate()
            out2 = out.split("uciok\n")[1]
            e = out2.split("\n")
            e.pop()
            e.pop()

            for element in e[:-1]:
                m = re.search('info depth\ (.+?)\ ', element)
                n = re.search('score\ (.+?)\ nodes', element)

                try:
                    output = output + m.group(1) + " " + n.group(1) + "\n"
                except:
                    pass

            output = output + e[-1] + "\n"

    return output
```

Código 5 - evaluate.py, evaluate_sf

Esta función recibe la matriz del punto anterior y evalúa movimiento a movimiento cada partida. Utiliza expresiones regulares para extraer exclusivamente la puntuación de cada movimiento.

La salida de esta función la recibe la función *analize* del archivo *analizer.py*. Esta función quedaría definida de la siguiente forma:

```
def analize(output):

    games = output.split("GAME")

    for a in games:

        if a:
            print "game " + a[0] + a[1]

        b = a.split("bestmove ")
        g = []
        for c in b:
            d = c.split("\n")
            g.append(d)

        counter = 0
        for h in g[:-1]:

            counter = counter + 1
            try:
                # Below lies the definition
                # of the problems that you seek
                re.search("mate",a)
                h1=int(h[1].split("cp ")[1])
                h7 = re.search("mate", h[7])
                if math.fabs(h1)<500 and h7:
                    print "PLY: " + str(counter)

            except IndexError:
                pass
```

Código 6 - analizer.py

Esta función es la encargada de comparar la evaluación a profundidad 7 y a profundidad 1. Si el resultado concuerda con los criterios establecidos de búsqueda de problemas (en este caso mate a profundidad 7 y ventaja de menos de 5 puntos a

profundidad 1) se devuelve por pantalla el movimiento en el que se encuentra el problema de táctica.

Puede parecer un error el hecho de que con esta implementación puedan encontrarse posiciones en las que sea el turno de blancas pero que las negras tengan mate en x movimientos. Pero no es un error ya que esto sería un problema de táctica, con la única condición de que se invierta el turno.

Por ejemplo, dada una posición que devuelve una puntuación de 100 a profundidad 1, y mate en -3 (mate en un número negativo significa mate para las negras) a profundidad 7, siendo turno de las blancas. Uno podría pensar que la posición no da lugar a un problema de táctica ya que es el turno de las blancas, y pase lo que pase les van a dar mate en 3. Sin embargo al declarar el turno de negras, esto se convierte un problema de mate para las negras.

4.5 Posibles mejoras

A continuación se describen algunas de las posibles mejoras aplicables a la solución aportada.

Una mejora que se podría dar en cuanto a eficiencia sería la de no calcular la evaluación de los primeros movimientos. Esto ahorraría un tiempo considerable ya que las posiciones iniciales son relativamente complejas y por lo tanto son computacionalmente más costosas que, por ejemplo, las de final de partida. Por otro lado no se perderían apenas resultados ya que las posiciones iniciales suelen ser repetidas. En cuanto al número de movimientos que evaluar, este sería objeto de estudio.

Otra posible mejora, sería la de mejorar la salida o representación de los problemas encontrados. Como el objetivo de este proyecto no es comercial ni mucho menos, este aspecto del programa no se ha tenido muy en consideración y, tal y como está desarrollado el código actualmente, la salida queda de la siguiente forma:

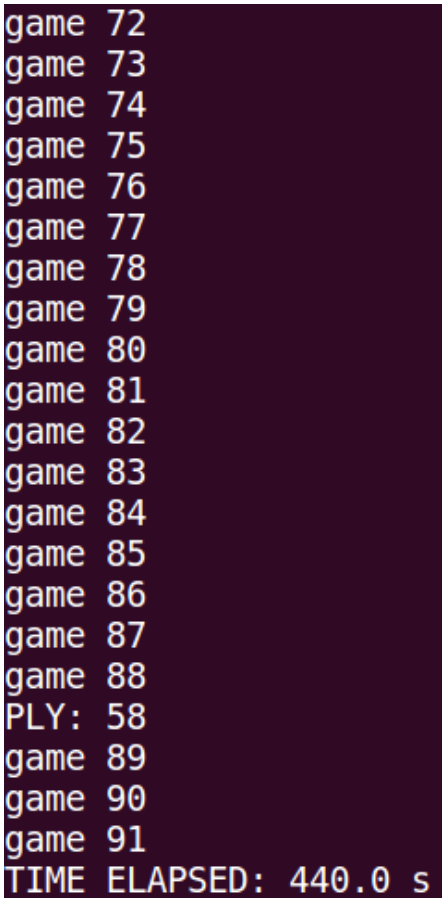

Ilustración 26- Output

Fuente: Elaboración propia

Se puede observar que se ha encontrado un problema de táctica en el medio movimiento (*ply* en inglés) 58 de la partida 88. Para el objetivo de este proyecto y su desarrollo esto no supone ningún problema pero una representación con una interfaz que incluyese un tablero supondría una notable mejoría aspectual.

Por último, otra posible mejora al proyecto sería añadirle la funcionalidad de poder establecer qué tipos de problemas se están buscando. Por ejemplo pasar por argumento un valor entre 2 y 4, donde ese número representa el número de movimientos para mate. La implementación de esta funcionalidad sería simplemente variar las profundidades a las que se compara en la función de búsqueda de problemas. Para mate en 2 sólo habría que encontrar mate a profundidad 3 y no mate a profundidad 1, para mate en 3 procedimiento análogo a profundidad 5 y 3, etc.

5. Pruebas

Se realizaron diversas pruebas tanto con la fórmula final como con fórmulas anteriores. A continuación se muestran algunos de los resultados y problemas encontrados por el programa. De un análisis de 10 partidas de jugadores aleatorios sin ELO se encontró el siguiente problema (mueven blancas) en un tiempo inferior a 1 minuto:

Ilustración 27 - Problema 3

Fuente: Elaboración propia mediante el editor de tableros de lichess.org

Este problema corresponde a la fórmula de análisis correspondiente a la diferencia en la evaluación a profundidad 5 y 1. Se probó esta fórmula sobre otras partidas pero los problemas encontrados resultaron ser demasiado sencillos y, por ello, se usó finalmente la fórmula explicada en la solución al problema.

De un torneo de ajedrez para menores de 10 años de Reikiavik (40 partidas) se sacaron estos dos problemas (mueven negras), utilizando la fórmula de profundidad 5:

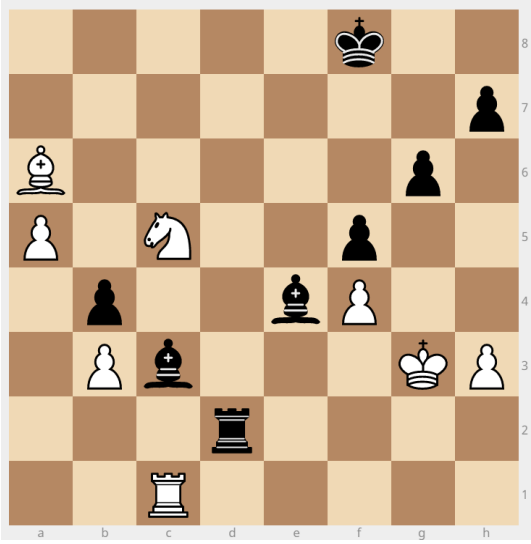

Ilustración 28 - Problema 4

Fuente: Elaboración propia mediante el editor de tableros de lichess.org

Ilustración 29 - Problema 5

Fuente: Elaboración propia mediante el editor de tableros de lichess.org

Estos tres primeros problemas todos son mates en 2, al poner profundidad 7 en la fórmula se encuentran mates hasta en 4 (profundidad 1 mates en 1 para blancas, profundidad 2 mates en 1 para las negras, profundidad 3 mates en 2 para las blancas, etc.).

Estos son dos problemas que se encontraron con la fórmula final, ambos mate en 4 para las blancas:

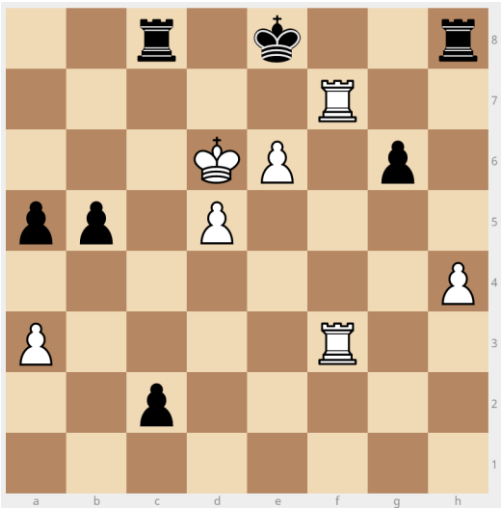

Ilustración 30 - Problema 6

Fuente: Elaboración propia mediante el editor de tableros de lichess.org

Ilustración 31 - Problema 7

Fuente: Elaboración propia mediante el editor de tableros de lichess.org

6. Marco regulador

Tanto el programa pgn-extract como el motor Stockfish están distribuidos bajo la licencia GNU[22] GPL (General Public License).

El sistema operativo GNU fue desarrollado para que los usuarios pudiesen tener libertad en sus tareas informáticas. En concreto el software libre implica que los usuarios tienen las 4 libertades esenciales:

- Ejecutar el programa
- Estudiar y modificar el código fuente del programa
- Redistribuir copias exactas
- Distribuir versiones modificadas

La filosofía del proyecto GNU se basa principalmente en que los usuarios de un programa de software deben ser libres de aprovechar sus posibilidades, no solo quien lo haya desarrollado.

Esta licencia permite el estudio, uso, copia y modificación del software. Por ello todo el código de este proyecto queda también distribuido bajo esta misma licencia.

[22] GNU: GNU's not Unix

7. Presupuesto económico y Cronograma

7.1 Presupuesto del proyecto

El presupuesto se ha calculado teniendo en cuenta que esta no es una aplicación destinada a uso comercial. Por ello, el presupuesto carece de costes de implantación, publicidad, registro, patente, etc.

Simplemente se reflejan los costes por la compra de material necesario (ordenador, licencia Office), los gastos de transporte (gasolina y abono transporte) para las reuniones físicas con el tutor, los gastos de impresión y libros.

Por otro lado, se calculan las horas de dedicación del tutor a un precio de 35€/hr y del alumno a un precio de 12€/hr.

Costes variables	Precio
Ordenador portátil Lenovo 310 -15IKB, I7-7500U	785,00€
Licencia Microsoft Office	19,75€
Otros Gastos asociados[23] al proyecto	85,00€
Total costes variables materiales	889,75€

Perfil	Coste/ hora	Horas	Total
Ingeniero Sr	35,00€	28	980,00€
Ingeniero Jr	12,00€	309	3.708,00€
Total coste variable personal [24]			4.688,00€

[23] Los gastos asociados incluyen gastos del transporte a las reuniones con el tutor, impresión de documento, material oficina, libros.

[24] No es necesario asumir costes de Seguridad Social ya que hasta 9.000€/año no es obligatorio. Tampoco está sujeto a IVA ya que es una actividad de investigación/docencia sin proyección comercial alguna y por tanto exento de IVA.

Coste variable material	889,75€
Coste variable personal	4.688,00€
Total coste	**5.577,75€**

7.2 Cronograma del proyecto

Desde el inicio del proyecto se ha ido anotando el tiempo dedicado a cada parte del mismo por lo cual el cronograma se ajusta a la realidad, a excepción del mes de octubre, la fase Defensa TFG (preparación) que es una previsión.

Fases del proyecto/Fechas	Abril	Mayo	Junio	Julio	Agosto	Septiembre	Octubre	horas
Selección idea/tutor								8
Diseño Estructura proyecto								31
Desarrollo TFG								106
Pruebas/Correcciones								86
Borradores/Revisiones								12
Remates/Entrega final								9
Defensa TFG (preparación)								3
Lecturas, bibliografía, búsquedas								54
							horas TFG	309

8. Conclusiones

La primera conclusión que se puede obtener en cuanto a la búsqueda de problemas de táctica en partidas de ajedrez es que la dificultad de la misma radica en la definición de aquello que conforma un problema. Es decir, dependiendo de los problemas que se estén buscando se utilizarán unas técnicas u otras.

Si por otro lado, el objetivo es encontrar cualquier tipo de problema de táctica, la estrategia consistiría en encontrar la fórmula adecuada para abordar el mayor número de problemas. La desventaja de tratar de encontrar el mayor número de problemas es que se encontrarán muchas posiciones que no tendrán mucho valor como problemas además de perderse algunas posiciones que quizás si lo tengan.

Otra conclusión a la que se llega desde el estudio de la historia de los problemas de táctica es la siguiente: diseñar o componer problemas no tiene nada en común con buscarlos en partidas, salvo el resultado.

La primera forma de generación de problemas probablemente dé lugar a problemas mejores en el ámbito estético. Por otro lado, la segunda forma permite generar muchos más problemas en menor tiempo, aunque quizás estéticamente no sean los mejores.

Otra ventaja de la búsqueda automática es la que se comentó en la introducción, ésta permite a un jugador encontrar problemas de sus propias partidas. Esta capacidad tiene un enorme valor para cualquier jugador ya que le permite aprender de sus errores.

La última conclusión obtenida tras la realización de este proyecto es que el campo de la búsqueda automática de problemas de táctica por ordenador está aún por explorarse. No hay prácticamente nada escrito sobre el tema, sólo se conoce su utilización por parte de portales de ajedrez. Creo que es un campo con un gran potencial, no sólo por su aplicación para encontrar problemas de táctica sino por todo lo que el estudio de las técnicas empleadas para su búsqueda puede conllevar.

Más allá de las conclusiones descritas explícitamente en este apartado, muchas se han ido describiendo durante todo el documento. Por ejemplo, las posibles mejoras del código incluyen aquellos aspectos que podrían suponer un avance futuro en este campo.

Abstract

This part of the document includes an English abstract describing the developed project.

Introduction

The motivation behind this project is the passion that the tutor and student have with the game of chess. The professor Carlos Linares proposed the project to me and since we both love this game I accepted it enthusiastically.

When it comes to the project itself, the real motivation behind it is to get better at the game. Like José Raúl Capablanca said: “You may learn much more from a game you lose than from a game you win. You will have to lose hundreds of games before becoming a good player”. Also it is known that tactic comes before strategy when it comes to learning chess.

With these thoughts in mind, the proposed objective is to build an automated chess tactic problem finder. The ultimate goal would be to pass a player’s set of games to the program so that it extracts the chess tactic problems from it.

For chess players this has huge value.

Summary

Before diving into chess tactics and the ways to find them, it is important to know some context and history surrounding chess. Since its origins in India chess has interested the people which it has come in contact with. The game is considered by many an art. It challenges the mind of those who play it and not only are they challenged by the player opposing them but also by the questions that the game poses to them.

Chess is a board game, for two players. The board consists of 64 squares on which 16 pieces are placed like this:

Ilustración 32 - Chessboard at the starting position

Source: The author's using Lichess' board editor

Each piece has its own strengths and weaknesses since every piece that has a different design moves differently. The objective of the game is to capture the opposing king, which is the piece represented by the crown of a king whose color is the opposite of yours. This objective is achieved by first eliminating the rest of the pieces of the enemy player and then with each move trapping the king until it cannot move and is being threatened by one of the pieces (check mate).

To understand what a chess tactic problem is first one must understand what tactic in chess means. Tactic in chess is the combination of moves that leads to a goal. Say for example that white wants to capture the black queen, then there might be some tactic consisting in a collection of moves that achieves this by trapping the queen.

A chess tactic problem is a position in which the player has to find the best move or tactic. This best move is defined as the one that gives the most advantage to the player that makes it. Here is an example of a chess tactic problem:

Ilustración 33 - Chess Tactic Problem 1

Source: The author's using Lichess' board editor

This chess tactic would say "White to move and mate in four". What this means is that white has to make four moves, and after the fourth one the black king will be check mated, and there is no way black can avoid the mate.

These chess tactic problems have been of interest almost since the origin of the game. At the beginning they were created manually by people. Amongst these people there is one that stands out, Samuel Lloyd.

Samuel Lloyd was an early prodigious not as a player of chess but as a creator of chess tactic problems. This serves as a proof that a player being a good chess tactic solver does not make him or her a good player.

But what is of real interest to this project is the automated finding of chess tactics. This is done by review of games and it has not been the focus of chess tactics enthusiasts until very recently. This is due to the fact that chess analysis requires high computation capability, both in memory and time.

Before getting into the techniques used to find problems of chess tactics in an automated manner first one has to know how a computer understands chess.

The way a computer or a program interacts with chess (nowadays) is by means of evaluating positions. This evaluation can be done at different depths; the deeper it goes the more moves it will be taking into account. So for example, evaluating at depth three means that the program will consider three moves in advance, that is one player moves, then the other one and then the first one again.

The evaluation is done by means of a search algorithm. The most used and probably the best algorithm for chess study is the Minimax algorithm with Alpha Beta pruning. This algorithm goes as follows: consider that white will take the best move for himself, and consider that black will take the best move for himself, both of them in terms of the evaluation function.

So, if for example, it is white's turn to move, and he has ten possible moves to make, the algorithm will "make" every single move and evaluate the position after each one of them. The best move will be the one that gives white the best evaluation, and if the search is done at depth one, the evaluation score will correspond to this move. If the algorithm were set to depth two, then after each move that white can make, each move that black could make, from the position after white's move, will also be executed. The evaluation returned will be the one after the move that maximizes white's score and the move that maximizes black's score (or minimizes white's score).

Let's take it a step further, considering depth three. In this case, the best evaluation for white might not be the one that returns the best evaluation for white after the first move. Since after this move black might have a good reply that sets its evaluation higher. There, what the algorithm does is to choose the moves that maximize white's evaluation after white moved, black replied and white moved again. So in a way, it will select the best two moves for white, considering that black will always make the best move.

Below these lines there is an example of what a search tree would look like for the Minimax algorithm applied to chess:

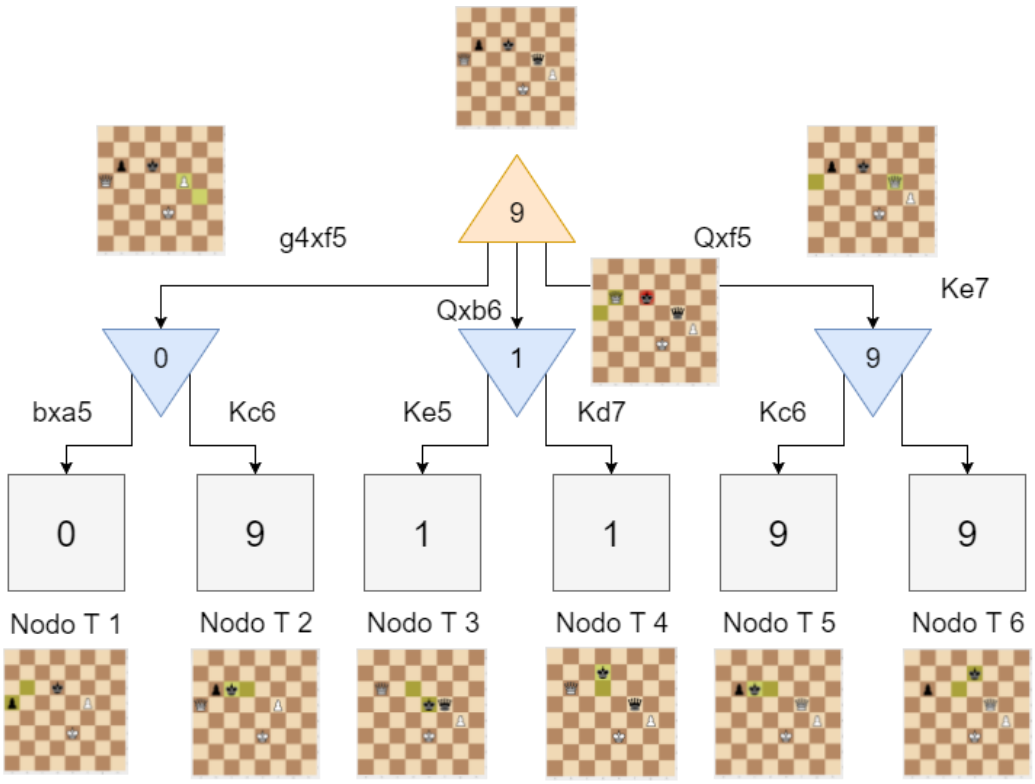

Ilustración 34 - Minimax Search Tree on Chess

Source: The author's using Lichess' board editor

As one can guess, this tree grows exponentially, that is where Alpha Beta pruning comes in. This improvement on the algorithm prunes the branches of the tree that can

no longer improve the score. That is done with evaluation score, if there is an evaluation score for the opponent that is lower than one found before in another branch, then the branch that is being explored is no longer explored. This is due to the fact that it would not, under any circumstance, improve the current score, since there is a move already that gives the opposing player a better answer than the one he or she already has for another branch.

One of the most important factors when applying these algorithms is designing a good evaluation formula. This will be the one determining whether a path is good or not, by means of evaluating the position. To give a quick understanding of the evaluation formula concept, a very simple one for chess would be to sum the value of the pieces for white and subtract to that the value of the pieces for black. This way, if in a given position white has a queen advantage, the evaluation formula would yield a value of nine. Where if, for example, black is up a knight, the evaluation would return negative three.

This is obviously a very simple formula, that would work only on a very basic level of chess, and it is in fact the formula that most of the beginner players use when they are playing their first matches: “¿How can I take the enemy pieces?”

Solution

Let’s now proceed to explain how the problem was approached from a theoretical and technical stand point.

In first place, from the theoretical point of view, a chess tactic problem as was defined before is a position where a player has to find the best move. So, the first thing to note here is that there has to be a good move to make, and one that is different on some degree to the rest of moves.

From the evaluation point of view, this means that on low depths, the evaluation might be pointing towards one direction, but that after the move that is the solution to

the problem is found the evaluation should shift drastically (to some point) into the other direction.

Consider the evaluation formula offered in the previous point. At the chess tactic original position, depth zero, the material might be perfectly balanced; therefore the evaluation would return a value of zero. But with two moves white can take black's queen and not lose a single piece in the process. Then at depth three, the evaluation would have a value of nine.

One can conclude that in a given position, if the low depth evaluation gives a value that differs greatly from the value that yields a deep evaluation, there is likely a tactic problem in that position.

In order to understand and grasp that concept what has been done in the early stages of this project was to analyze, at different depths, positions from tactical problems. The result of these analyses was precisely this; a great difference in the evaluation at different depths of the same position.

Secondly, from the technical stand point there are two programs that were used to develop this project. These are: PGN-Extract and Stockfish.

PGN-Extract is a program developed by a professor at the University of Kent, which serves the purpose of translating chess games from one format to another. This has proven to be very useful since the other program that is used, Stockfish, only accepts two formats, which are not the most commonly used PGN format.

Stockfish is a chess engine. This program gives evaluation up to the depth that the computer that it runs on can handle. The main key for the success of this program is its evaluation formula, which takes into account numerous factors each playing a part in the game. Its success is proved by the fact that it can actually beat the best players in the world.

The easiest way to use the Stockfish engine is by using the UCI protocol, a protocol designed to universalize chess engines so that they can interact with each other and be used more easily. The UCI protocol takes games in the UCI format, that is where PGN-Extract comes in, it is able to translate games from the widely used PGN format (or any other format) to the chess engine understandable UCI format.

In order to automate the process of finding chess tactic problems, the developed program has to take a large number of games and process them move by move, analyzing every position, evaluating at different depths to check whether there is a huge difference in the value returned at these different depths.

The process starts by retrieving the games that one wants to process; the ideal situation would be to have a set of games from a player. The problems that would get extracted out of these games would be of special utility to the player whose games they come from.

Once the games have been retrieved, usually in PGN format, they have to be translated into the UCI format, so that they can be understood by Stockfish.

After the games are translated into UCI they are split into every move. Therefore, the program obtains a matrix in which every row is a game and every column a move from that game. With that matrix, every single position is passed onto evaluation at a certain depth.

Finally, with the result from the evaluation, the program compares the evaluation at a low depth with that of a high depth from the same position. If the evaluation differs greatly, then it returns the move and the game where the position is to be found, so that the player can trace the position where there is a tactic problem.

There were several formulas tried during the development of the project, the one that finally yielded best results was to compare the evaluation at depth 1, and at depth 7. But in this case there is not a quantitative difference analysis; there is a simpler approach since for the sake of simplicity the program only finds mate problems.

Therefore, the program searches for positions where at depth 1 there is an advantage no greater than 500 centipawns[25] and at depth 7 there is a mate. This way it will find problems where apparently there is no great advantage to any of the players but that with the right set of moves (solution to the chess tactic problem) there is a mate in a certain number of moves.

This is one of the problems that were found with the program:

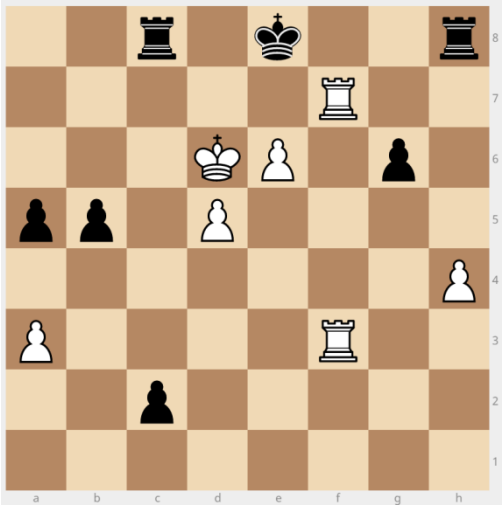

Ilustración 35 - Chess Tactic Problem 2

Source: The author's using Lichess' board editor

[25] Centipawn: Unit of measure of a player's advantage in a given position. 100 centipwans are equivalent to a pawn advantage.

Conclusions

There is a conclusion that became clear since the beginning of the development of the project. This is the fact that the difficulty of finding chess tactic problems is directly related to the types of problems that one seeks. This means that the technique and formula applied to searching problems will depend on these afore mentioned types of problems.

If, on the other hand, the objective is to find the maximum number of problems the strategy would be to try to cover the vast majority of problems. This is done by defining a formula that does not neglect any problem. The disadvantage with this approach is that it will most likely find many positions that are not considered chess tactic problems.

A forgone conclusion that was reached during the investigation phase of the project is that designing a chess tactic problem has nothing to do with finding or automatically extracting problems from chess games. The only aspect that they share is the final result. Although the first technique will most likely yield better problems, or esthetically more refined, the second one is capable of creating or mining a greater number of problems in lesser time.

As it was said before, another advantage of using automated search of chess tactic problems is that a player can find problems of his or her own games. This holds extraordinary value for those of us who want to improve our game.

There is one last conclusion regarding automated search of chess tactic problems. This is that there has not been a thorough investigation of the matter; in fact there are not many (if any) papers or articles written on it. The entities that have done the most intensive research on it are the chess portals that use this technique to get tactic problems for their users, such as *lichess.org* and *chesstempo.com.*

Anexos

I. Definiciones

Ajedrez para 3: Modalidad de juego del ajedrez en el que tablero ha sido modificado para que jueguen 3 jugadores, todos contra todos.

Alan Mathison Turing: (Londres 1912 – Cheshire 1954), es considerado el padre de la computación e informática moderna.

Bughouse: Modalidad de juego del ajedrez en el que juegan 4 jugadores, 2 contra 2. Las piezas que se toman en un tablero se pueden colocar utilizando un turno en el otro tablero.

Centipeón: Unidad de medida de la ventaja de un jugador en una posición determinada. 100 centipeones equivalen a un peón de ventaja.

Columna abierta: En ajedrez es aquella columna que no tiene peones.

Columna semi-abierta: En ajedrez es aquella columna que sólo tiene un peón.

Gran Maestro: Título de mayor categoría de ajedrez. Lo adquieren aquellos jugadores que demuestran ser expertos en el juego de ajedrez.

Johann Wolfgang Ritter von Kempelen de Pázmánd: (Bratislava 1734 – Viena 1804) Noble y escolar húngaro, gran jugador de ajedrez que acostumbraba a jugar con la nobleza austrohúngara. Es conocido por ser el creador del Turno Mecánico.

John Von Neumann: (Budapest 1902 – Washington 1957), es considerado uno de los matemáticos más importantes de la era moderna por la importancia de sus trabajos en múltiples áreas.

José Raúl Capablanca: (La Habana, Cuba 1888- Nueva York, EEUU 1942) Ajedrecista cubano campeón mundial de ajedrez desde 1921 a 1927. Es un referente de ajedrez a nivel mundial.

Juego de suma cero: Aquellos juegos en los que la ganancia o pérdida por parte de un jugador es contrarrestada por la ganancia o perdida de los demás jugadores.

Magnus Carlsen: (Akershus, Noruega 1990) Ajedrecista noruego, segundo campeón mundial más joven de la historia. Es el actual campeón del mundo y se sitúa en el puesto número 1 en el ranking por ELO con 2826 puntos.

Mahabharata: Obra de literatura india que data del siglo III a.C de carácter bélico y mitológico.

Outpost: En ajedrez se da cuando existe una pieza menor protegida por un peón en una posición en la que la pieza menor no puede ser atacada por un peón.

Rating ELO: Sistema de puntuación estadístico que mide la fuerza relativa de los competidores.

Savielly Tartakower: (Rostov del Don 1887, Rusia París, Francia 1956) Ajedrecista polaco-ruso, uno de los fundadores de la corriente híper-moderna del ajedrez.

Stockfish: motor de ajedrez UCI (Interfaz de ajedrez Universal) de código abierto.

II. Siglas y Acrónimos

FEN: acrónimo de Forsyth-Edwards Notation (Notación Forsyth-Edwards)

GNU: GNU's Not Unix (GNU No Es Unix)

PGN: acrónimo de Portable Game Notation (Notación portátil de partida.

UCI: acrónimo de Universal Chess Interface (Interfaz Universal de Ajedrez)

WFCC: World Federation of Chess Composition (Federación Mundial de Comosición Ajedrecística).

III. Soluciones a los problemas

Problema 1: 1. Nf5+ Ke8 2. Nxg7#

Problema 2: 1. b4 Rc5+ 2. bxc5 a2 3. c6 Bc7 4. cxb7 any 5. bxa8=Q#

Problema 3: 1. Qg4 Qg1+ 2. Kxg1 any 3. Qg8# (2 … Bg7 3. Qxg7#)

Problema 4: 1. … Rg2+ 2. Kh4 Bf6#

Problema 5: 1. … Qh3+ 2. Qh2 Re1#

Problema 6: 1. e7 Rc6+ 2. dxc6 any 3. Rg8+ Rxg8 4. Rg8#

Problema 7: 1. Rxf8+ Kxf8 2. Qc8+ Rd8 (2. … Ke7 3. Qe8#) 3. Qxd8+ Qe8 4. Qxe8#

Bibliografía

[1] Henry E. Bird. *Chess history and reminiscences*, 10th Edition. Project Gutenberg. 2004. **[En línea]. Disponible en:** http://www.gutenberg.org/ebooks/4902

[2] Stewart Gordon. "The game of kings", *Aramco World*. Volumen 60, número 40, páginas 18-23. Agosto-2009. **[En línea]. Disponible en:** http://archive.aramcoworld.com/issue/200904/the.game.of.kings.htm Acceso: Julio-2017

[3] Y. Averbakh y G. Kasparov, *A History of Chess: from Chaturanga to the present day,* 1st Edition. USA. Russell Enterprises, Inc, 2012.

[4] Irving Chernev, *The Most Instructive Games of Chess Ever Played*, 1st Edition. New York, USA. Dover, 1992.

[5] J. J. O'Connor y E. F. Robertson. "Lloyd Biograpy"*, University of Saint Andrews, Scotland. History of Mathematics archive.* Octubre-2003. **[En línea]. Disponible en:** http://www-history.mcs.st-andrews.ac.uk/Biographies/Loyd.html

[6] A. Soltis, *Sam Loyd: His story and best problems,* 1st Edition. USA. Chess Digest, 1995

[7] "The World Federation for Chess Composition", *The World Federation for Chess Composition.* Mayo-2013. **[En línea]. Disponible en:** http://www.wfcc.ch

[8] Tibor Karolyi y Nick Aplin, *Kasparov's Fighting Chess 1999-2005*, 1st Edition. United Kingdom. Batsford Chess, 2006.

[9] Leontxo García, *Ajedrez y ciencia pasiones mezcladas,* Primera Edición. España. Editorial Crítica, 2013.

[10] "History of Computer Chess", *Chess Programming Wiki*. 2017. **[En línea]. Disponible en:** https://chessprogramming.wikispaces.com/History

[11] "Lichess", *Lichess*. 2017. **[En línea]. Disponible en:** https://lichess.org/

[12] "Chess Tempo", *Chess Tempo*. 2017. **[En línea]. Disponible en:** https://chesstempo.com/

[13] Ahmad Abdolsaheb. "How to make your Tic Tac Toe game unbeatable by using the minimax algorithm", *Medium*. 18-02-2017. **[En línea]. Disponible en:**

https://medium.freecodecamp.org/how-to-make-your-tic-tac-toe-game-unbeatable-by-using-the-minimax-algorithm-9d690bad4b37

[14] Eric Mayefsky, Francine Anene y Marina Sirota. “Algorithms – Minimax”, *University of Standford.* 2003 **[En línea]. Disponible en:** https://cs.stanford.edu/people/eroberts/courses/soco/projects/2003-04/intelligent-search/minimax.html

[15] Bruce Rosen. “Computer Science 161 Recitation Notes, Minimax”, *University of Stanford.* Septiembre 2009. **[En línea]. Disponible en:** http://web.cs.ucla.edu/~rosen/161/notes/minimax.html

[16] Eric Mayefsky, Francine Anene y Marina Sirota. “Algorithms – AlphaBeta”, *University of Standford.* 2003 **[En línea]. Disponible en:** https://cs.stanford.edu/people/eroberts/courses/soco/projects/2003-04/intelligent-search/alphabeta.html

[17] Bruce Rosen. “Computer Science 161 Recitation Notes, Minimax with Alpha Beta Pruning”, *University of Stanford.* Septiembre 2009. **[En línea]. Disponible en:** http://web.cs.ucla.edu/~rosen/161/notes/alphabeta.html

[18] “Stockfish Evaluation Guide”, *Stockfish Evaluation Guide.* Septiembre 2017. **[En línea]. Disponible en:** https://hxim.github.io/Stockfish-Evaluation-Guide/

[19] “Stockfish Source Code”, *Official Stockfish*. Septiembre 2017. **[En línea]. Disponible en:** https://github.com/official-stockfish/Stockfish

[20] “Stockfish Home Page”, *Stockfish*. 2017. **[En línea]. Disponible en:** https://stockfishchess.org/

[21] David J. Barnes. “PGN-Extract”, *PGN-Extract*. Septiembre 2017. **[En línea]. Disponible en:** https://www.cs.kent.ac.uk/people/staff/djb/pgn-extract/

[22] Stefan-Meyer Kahlen. “Universal Chess Interface”, *Universal Chess Interface*. Abril 2004. **[En línea]. Disponible en:** http://wbec-ridderkerk.nl/html/UCIProtocol.html