UNIVERSIDAD DE ALMERÍA

ESCUELA POLITÉCNICA SUPERIOR

Departamento de Estadística y Matemática Aplicada

BAYESCHESS: PROGRAMA DE AJEDREZ ADAPTATIVO BASADO EN REDES BAYESIANAS

PROYECTO FIN DE CARRERA

Ingeniero en Informática

Antonio Fernández Álvarez

Director: Antonio Salmerón Cerdán

Junio de 2005

A mis padres, por su esfuerzo constante

Agradecimientos

Quiero expresar mis agradecimientos a todas las personas que me han apoyado en la elaboración del presente proyecto, así como durante todos los años de carrera.

En primer lugar, a mi director de proyecto Antonio Salmerón, por su ayuda y su actitud acogedora durante este período.

A Moisés, por sus consejos sobre LaTeX mientra desarrollaba la memoria del proyecto.

A José Manuel, por haber hecho más fácil mis inicios en Java.

A Julio, por su apoyo y sabios consejos en los primeros años de carrera.

A Ángel, por su presencia junto a mí durante interminables días en la biblioteca de la Universidad.

A Charo, por respetar mis momentos de estudio y por animarme en las situaciones difíciles.

A Paco Maldonado, por sus explicaciones sobre ajedrez y por prestarme su ordenador para la generación de la base de datos de partidas. También, a su mujer Charo por esas cenas improvisadas.

A los *polis* Ángel y Paco por hacer inolvidables dos años de convivencia y estudio en común.

Y, por último, y muy especialmente, a mis padres y a mi hermano, por su constante generosidad durante tantos años fuera de casa.

Índice general

Índice de tablas

Índice de figuras

Parte I

Memoria

Capítulo 1

Introducción

1.1. Estructura del documento

El documento se divide en dos partes:

- I: Memoria
- II: Apéndice

En la Memoria, se englogan 5 capítulos que contienen la parte más importante del documento, mientras que en el Apéndice incluiremos algunos aspectos relacionados con el proyecto, que considero de menor importancia.

En cuanto a la parte I, el primer capítulo *Introducción* contiene el anteproyecto, que engloba los objetivos, la metodología y planificación a seguir, y una primera aproximación a la bibliografía necesaria. También incluye un apartado de localización para situar y delimitar el proyecto y sus objetivos en un mundo tan variado y cambiante como la Informática.

En el capítulo 2, *Fundamentación Teórica*, se incluye toda la base teórica de cada una de las técnicas necesarias para la implementación de la aplicación. En concreto se habla de la búsqueda en problemas de juegos, modelos gráficos probabilísticos y técnicas de clasificación basadas en redes Bayesianas.

En el capítulo 3, *Diseño del motor de juego de ajedrez*, se detalla todo el proceso de diseño del juego de ajedrez, sin incluir todavía aspectos de aprendizaje y adaptación. Estableceremos un punto de partida a la hora de comenzar el diseño e implementación, plantearemos una heurística de juego inicial y explicaremos el formato del árbol de búsqueda con el que juega el ajedrez. Por último se comentarán algunas de las situaciones especiales en ajedrez que han debido ser programadas (jaque mate, tablas, ...).

En el capítulo 4, *Aprendizaje automático*, se diseña un clasificador para el aprendizaje, se especifica cada una de las variables que lo forman, con sus posibles valores y se explicará cómo se realiza el proceso de aprendizaje de la red Bayesiana a partir de una base de datos de partidas.

En el capítulo 5, *Adaptación a la situación del oponente*, se explicarán los tipos de estrategias de juego que existen en el mundo del ajedrez y la técnica que hemos desarrollado para adaptarnos a ellas.

En cuanto al *Apéndice*, el primer capítulo se ha destinado a explicar la *Notación algebraica en ajedrez*, ya que ha sido la que se utiliza para llevar un registro de las partidas y leerlas en la generación de la base de datos de entrenamiento del clasificador de estrategia.

En el segundo capítulo, se han tratado algunos *Aspectos de diseño e implementación* de la aplicación. Se ha incluído un diagrama de clases de la aplicación que nos dará una visión un poco más proxima a la implementación, y se han explicado algunas estructuras de datos utilizadas, y que considero de importancia.

Por último, en el tercer capítulo dedicado al *Manual de usuario*, se han desglosado las distintas opciones del programa y explicado para que sirve cada una de ellas.

1.2. Anteproyecto

1.2.1. Título

"*BAYESCHESS*: PROGRAMA DE AJEDREZ ADAPTATIVO BASADO EN REDES BAYESIANAS"

1.2.2. Introducción

Las redes bayesianas se han convertido en una potente herramienta para la construcción de sistemas inteligentes capaces de adaptarse ante la llegada de nueva información. Una de las últimas aplicaciones de las mismas radica en la elaboración de sistemas adaptables al usuario, como pueden ser los filtros de correo electrónico y los sistemas de recomendación de compras en portales de venta por internet.

En el caso del ajedrez por computador, problema basado en la búsqueda en un espacio de soluciones guiado por una heurística, la interacción con el usuario se da en forma de juego, donde las decisiones de la máquina se van refinando en función de los resultados obtenidos en las partidas jugadas contra los diferentes usuarios.

1.2.3. Objetivos

El objetivo del proyecto es el diseño de un programa de ajedrez basado en redes bayesianas y con capacidad adaptativa desde dos puntos de vista:

1. La función heurística se refinará en base a los resultados de las partidas jugadas.

2. El programa determinará su forma de juego dependiendo del usuario al que se enfrente.

1.2.4. Metodología

El software a desarrollar puede descomponerse en dos bloques fundamentales: el motor de ajedrez y el componente adaptativo basado en redes bayesianas.

En cuanto al motor de ajedrez, el alumno podrá recurrir a software libre a partir del cual construir el sistema de visualización del tablero y de validación de movimientos. A continuación, se implementará el núcleo del juego usando un método de búsqueda aplicado a juegos, tipo MINIMAX con poda ALFA-BETA, cuya pertinencia quede debidamente acreditada.

Posteriormente se elegirán varias heurísticas de búsqueda que se correspondan a distintos patrones de actuación por parte del programa (llamados estilos de juego en el mundo ajedrecístico: agresivo, posicional, ...).

En cuanto al componente adaptativo, se construirá una red bayesiana que permita reponderar los componentes de la función heurística en base a los resultados obtenidos. Por otro lado, se construirá otra red bayesiana que, ante la forma de jugar del usuario, lo clasifique dentro de un patrón, y a partir de ese patrón la máquina decida el estilo de juego que adoptará.

Este componente adaptativo resulta novedoso y debe constituir la parte fundamental del proyecto.

Para el diseño de las redes bayesianas podrá usarse el software Elvira si el alumno lo estima oportuno. Algunas funciones de su API podrán ser integradas en el programa BayesChess para la realización de tareas de manipulación de las redes. Se implementarán las funciones que sean necesarias.

La implementación del proyecto será realizada en el lenguaje de programación JAVA, versión 1.4.2 para garantizar la compatibilidad con la API de Elvira.

1.2.5. Planificación a seguir

1. Revisión de conocimientos previos: El alumno deberá ponerse al día en las materias cursadas en la titulación relacionadas con los problemas de búsqueda, juegos,

técnicas de inferencia probabilística y programación en Java.

2. Seguidamente deberá realizar una descomposición funcional de cada una de las partes en las que se divide en proyecto y se especificará el diseño del software necesario.

3. A continuación se pasará a la fase de implementación con las restricciones impuestas en el apartado de metodología.

4. Se continuará con una fase de verificación del software y refinamiento del módulo adaptativo.

5. Todas las fases del proyecto se documentarán debidamente constituyendo la memoria final del proyecto.

1.2.6. Bibliografía básica

1. Castillo, E., Gutiérrez, J.M., Hadi, A.S. Sistemas expertos y modelos de redes probabilísticas. Monografías de la Academia de Ingeniería. 1996.

2. Fürnkranz, J.. Machine Learning In Computer Chess: The Next Generation. International Computer Chess Association Journal 1996.

3. Gámez, J.A. Puerta, J.M. Sistemas expertos probabilísticos. Colección Ciencia y Técnica. Ediciones de la Universidad de Castilla- La Mancha. 1998

4. Jensen, F.V. An introduction to Bayesian networks. UCL Press. 1996

5. Nilsson, N. Inteligencia artificial: una nueva síntesis. McGraw Hill, 2004

1.3. Localización

Es importante delimitar claramente este proyecto para situarnos y saber de antemano por donde nos estamos moviendo.

El presente proyecto pertenece a la carrera *Ingeniero en Informática* impartida en la *Universidad de Almería* y se engloba dentro del perfil de *Inteligencia Artificial*. Dentro de éste, se clasifica en un ámbito que está en auge actualmente, las redes Bayesianas. En

concreto, en el aprendizaje y adaptación al entorno usando redes Bayesianas.

Existen algunas asignaturas en la carrera relacionadas directa o indirectamente con este tema y con algunos aspectos necesarios para el desarrollo de este proyecto. Por citar algunas: *Técnicas de Inferencia Probabilística*, *Fundamentos de Inteligencia Artificial y Sistemas Expertos*, *Inteligencia Artificial e Ingeniería del Conocimiento*, *Estadística*, *Ingeniería del Software*, *Teoría de Grafos*, ...

El primer contacto que tuve con las redes Bayesianas fue en la asignatura Técnicas de Inferencia Probabilística, impartida por el profesor Dr. Antonio Salmerón Cerdán, director de mi proyecto. Posteriormente, en un curso, denominado "Técnicas Estadísticas aplicadas al análisis de datos y su tratamiento informático", pude trabajar en este tema también. A partir de aquí aumentó mi interés en este ámbito y comencé a pensar en posibles proyectos fin de carrera relacionados, y por lo que se ve, ya he cumplido mi deseo.

Capítulo 2

Fundamentación teórica

2.1. Búsqueda en problemas de juegos

Los entornos competitivos, en los cuales los objetivos de dos o más partes están en conflicto, dan lugar a los problemas de búsqueda entre adversarios, a menudo conocido como juegos. Éstos provocan una inexplicable fascinación y la idea de que las computadoras puedan jugar existe desde su creación:

- Siglo XIX, Babbage, arquitecto de computadoras, pensó en programar su máquina analítica para que jugara al ajedrez.
- '50, Shannon describió los mecanismos que podían usarse en un programa que jugara al ajedrez.
- '50, Turing describió un programa para jugar al ajedrez pero no lo construyó.
- '60, Samuel construyó el primer programa de juegos importante y operativo, el cual jugaba a las damas y podía aprender de sus errores para mejorar su comportamiento.

Los juegos proporcionan una tarea estructurada en la que es muy fácil medir el éxito o el fracaso. En comparación con otras aplicaciones de inteligencia artificial, por ejemplo comprensión del lenguaje, los juegos no necesitan grandes cantidades de conocimiento.

En Inteligencia Artificial los juegos son tratados en un entorno limitado, en donde las acciones de dos agentes se alternan y los valores de utilidad suman cero. Esto es, durante el juego la ventaja de un jugador sobre su adversario es igual a la desventaja de éste y al

final de juego, son siempre iguales y opuestos. Por ejemplo, en el caso del ajedrez, si un jugador gana (+1), el otro necesariamente debe perder (-1); si hay tablas ambos jugadores obtienen la misma utilidad (0).

Dentro de la gama de juegos, que durante la historia han sido foco de atención de la Inteligencia Artificial, se encuentra el ajedrez, debido a su dificultad para resolver situaciones. El ajedrez tiene un factor de ramificación promedio de aproximadamente 35, y los juegos van a menudo a 50 movimientos por cada jugador, por tanto el árbol completo de búsqueda tiene aproximadamente 35^{100} o 10^{154} nodos. Los juegos, como en el mundo real, requieren la capacidad de tomar alguna decisión cuando es infactible calcular la decisión óptima. Por ello, la investigación en este tema nos ha aportado algunas ideas interesantes sobre cómo hacer uso, lo mejor posible, del tiempo.

Para obtener un movimiento se explora en un árbol de búsqueda para encontrar la mejor solución. Una de las técnicas adoptadas con frecuencia es la poda, que nos permite ignorar partes del árbol de búsqueda que no marcan ninguna diferencia para obtener la opción final. Por otro lado la función de evaluación nos permite obtener una aproximación de la utilidad verdadera de un estado sin hacer una búsqueda completa.

2.1.1. Búsqueda en profundidad

Un recorrido en profundidad (en inglés DFS - Depth First Search) es un algoritmo que permite recorrer todos los nodos de un grafo o árbol de manera ordenada, pero no uniforme. Su manera de funcionar se basa en ir expandiendo cada una de los nodos que va localizando, de manera recursiva, recorriendo todos los nodos de un camino concreto. Cuando ya no quedan más nodos por visitar en este camino o se haya alcanzado un límite de profundidad fijado previamente (búsqueda en profundidad limitada), regresa hacia atrás (backtracking), de tal manera que comienza el mismo proceso con cada uno de los hermanos del nodo ya procesado. En concreto, la búsqueda en profundidad limitada es la que utiliza el algoritmo MINIMAX que trataremos posteriormente. Análogamente existe el algoritmo de búsqueda en anchura (BFS - Breadth First Search), no necesario para este proyecto.

2.1.2. Algoritmo minimax

A partir de ahora limitaremos los juegos a dos jugadores, que denominaremos MAX y MIN, por motivos que pronto se harán evidentes. MAX es el jugador que mueve primero, y luego mueven por turno hasta que el juego se termina. Al final, se conceden puntos al ganador y penalizaciones al perdedor.

Un juego puede definirse formalmente como una clase de problemas de búsqueda con los componentes siguientes:

- El **estado inicial**, que incluye la posición del tablero e identifica al jugador que mueve
- Una **función sucesor**, que devuelve una lista de pares (*movimiento, estado*), indicando un movimiento legal y el estado que resulta
- Un **test terminal**, que determina cuándo se termina el juego. A los estados donde el juego se ha terminado se les llaman **estados terminales**
- Una **función utilidad** (también llamada función objetivo o de rentabilidad), que da un valor númerico a los estados terminales. En el ajedrez, el resultado es triunfo, pérdida o empate (1, -1, 0, respectivamente)

El estado inicial y los movimientos legales para cada jugador forman el árbol de juegos. En la Figura 2.1 vemos parte del árbol de búsqueda para el caso del juego *tres en raya*.

Desde el estado inicial, MAX tiene nueve movimientos posibles. El juego alterna entre la colocación de una X para MAX y la colocación de una O para MIN, hasta que se alcance los nodos hoja correspondientes a estados terminales, en los que un jugador tiene tres en raya o todos los cuadrados están llenos. El número sobre cada nodo hoja indica el valor de utilidad del estado terminal desde el punto de vista de MAX; se supone que los valores altos son buenos para MAX y malos para MIN (de ahí el nombre de los jugadores).

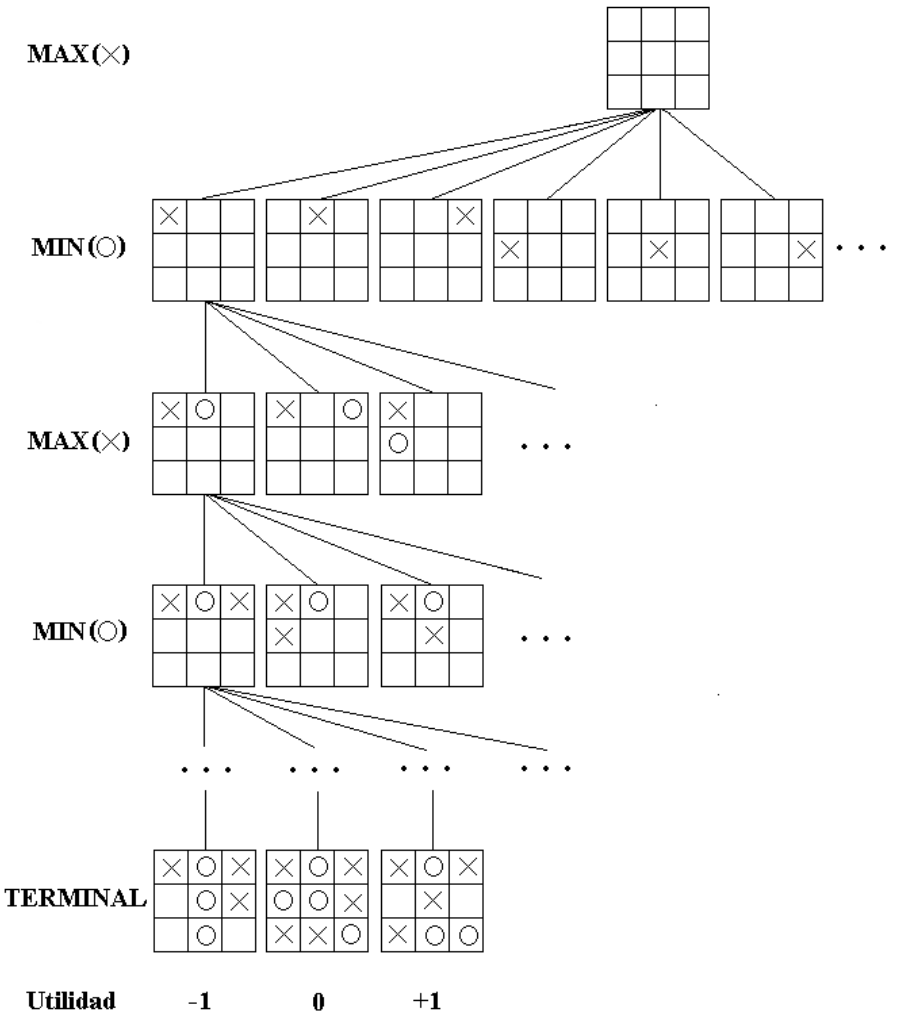

Figura 2.1: Árbol de búsqueda en el juego del tres en raya

Vemos como en un juego tan simple ya es difícil dibujar el árbol de búsqueda completo.

Considerando un árbol de juegos, la estrategia óptima puede determinarse examinando el **valor minimax** de cada nodo, que denotamos como el VALOR-MINIMAX(n). El valor minimax de un nodo es la utilidad (para MAX) de estar en el estado correspondiente *asumiendo que ambos jugadores juegan óptimamente* desde allí hasta el final del juego. En la Figura 2.2 se muestra cómo se calcula el VALOR MINIMAX de un nodo del árbol.

$$VALOR-MINIMAX(n) = \begin{cases} UTILIDAD(n) & \text{si n es terminal} \\ max_{s \in Sucesores(n)} VALOR-MINIMAX(s) & \text{si n es MAX} \\ min_{s \in Sucesores(n)} VALOR-MINIMAX(s) & \text{si n es MIN} \end{cases} \quad (2.1)$$

Figura 2.2: VALOR MINIMAX de un nodo

Apliquemos estas definiciones al árbol de juegos de la Figura 2.3. Los nodos terminales se etiquetan con sus valores de utilidad. A partir de aquí cada uno de los restantes nodos se etiquetan con el valor minimax desde las hojas a la raíz. Al final del proceso la **decisión minimax** la identificamos en la raíz. La primera alternativa con utilidad 12 es la opción óptima para MAX porque conduce al sucesor con el valor minimax más alto.

En la Figura 2.4 podemos ver cómo funciona el algoritmo minimax con todo detalle. Lo que hace es calcular la decisión minimax del estado actual. Usa un cálculo simple recurrente de los valores minimax de cada estado sucesor, directamente implementando las ecuaciones de la definición de la Figura 2.2. La recursión avanza hacia las hojas del árbol, y entonces los valores minimax **retroceden** por el árbol cuando la recursión se va deshaciendo. Por ejemplo en el árbol de la Figura 2.3, el algoritmo primero va hacia abajo a los cuatro nodos izquierdos, y utiliza la función de utilidad para descubrir que sus valores son 12, 4, 4 y 3 respectivamente. Entonces toma el máximo (puesto que nos encontramos en un estado MAX) de estos valores, 12, y actualiza la utilidad del nodo actual con dicho valor. Un proceso similar devuelve el valor 28, 8, 29, 11, y 34 restantes cinco nodos del mismo nivel. A continuación se repite el proceso en el nivel superior correspondiente a un estado MIN. Los valores obtenidos son 12, 8 y 11. Por último, la decisión minimax es el

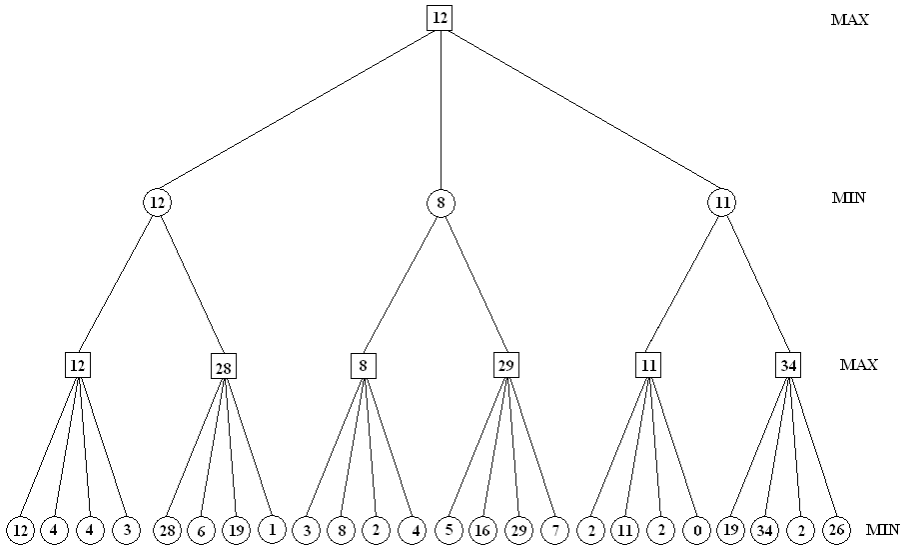

Figura 2.3: Obtención del mejor movimiento usando minimax

máximo de estos dos valores, esto es, 12, tal y como apuntábamos anteriormente.

Esta definición de juego óptimo para MAX supone que MIN juega también óptimamente (maximiza los resultados del *caso-peor* para MAX). ¿Y si MIN no juega óptimamente? Entonces es fácil demostrar que MAX lo hará mucho mejor.

El algoritmo minimax realiza una exploración primero en profundidad completa del árbol de juegos. Si la profundidad máxima del arbol es m, y hay b movimientos legales en cada punto, entonces la complejidad en tiempo del algoritmo minimax es del $O(b^m)$. La complejidad en espacio es $O(bm)$ para un algoritmo que genere todos los sucesores a la vez, o $O(m)$ para un algoritmo que genere los sucesores uno por uno. Para juegos reales, desde luego, los costos de tiempo son totalmente poco prácticos, pero este algoritmo sirve como base para el análisis matemático de juegos y para algoritmos más eficientes computacionalmente, por ejemplo la poda alfa-beta, que será explicada a continuación.

2.1.3. Poda alfa-beta

Para el caso del juego de ajedrez resulta casi inabordable usar el algoritmo minimax sin ninguna mejora, ya que el número de estados que tiene que examinar es exponencial en

```
función DECISIÓN-MINIMAX(estado) devuelve una acción
    variables de entrada:
        estado, estado actual del juego

    v ← MAX-VALOR(estado)
    devolver la acción de SUCESORES(estado) con valor v
```

```
función MAX-VALOR(estado) devuelve un valor utilidad
    variables de entrada:
        estado, estado actual del juego

    si TEST-TERMINAL(estado) entonces
        devolver UTILIDAD(estado)
    v ← −∞
    para s en SUCESORES(estado) hacer
        v ← MAX(v, MIN-VALOR(s)
    devolver v
```

```
función MIN-VALOR(estado) devuelve un valor utilidad
    variables de entrada:
        estado, estado actual del juego

    si TEST-TERMINAL(estado) entonces
        devolver UTILIDAD(estado)
    v ← +∞
    para s en SUCESORES(estado) hacer
        v ← MIN(v, MAX-VALOR(s)
    devolver v
```

Figura 2.4: Algoritmo minimax

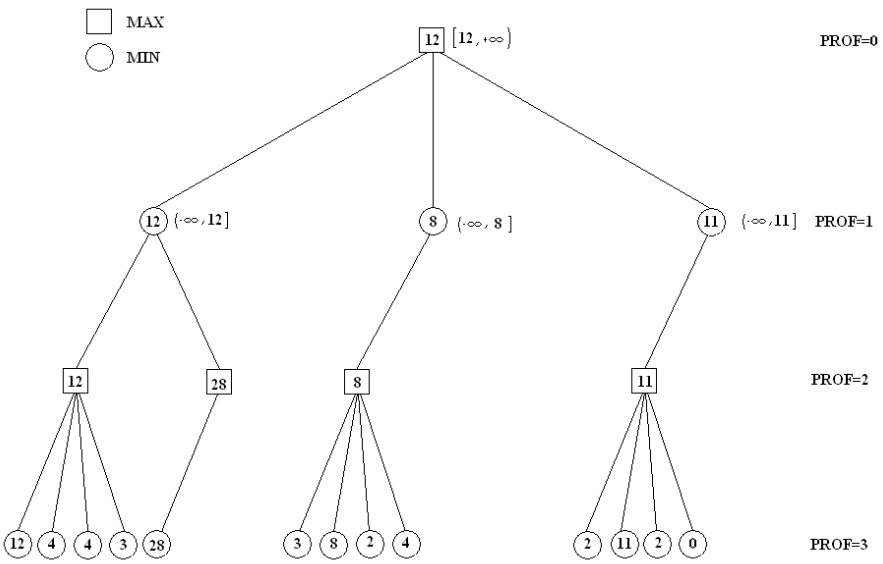

Figura 2.5: Obtención del mejor movimiento usando poda alfa-beta

el número de movimientos (nivel de profundidad que se explore en el árbol de búsqueda). Lamentablemente no se puede eliminar el exponente en la complejidad del algoritmo minimax, pero podemos dividirlo, incluso hasta la mitad. Lo que plantea la **poda alfa-beta** es que es posible calcular la decisión minimax correcta sin mirar todos los nodos en el árbol de búsqueda. Es decir podemos **podar** o eliminar partes del árbol que no influyen en la decisión minimax final.

La poda alfa-beta recibe su nombre de los dos parámetros que describen los límites sobre los valores hacia atrás que aparecen a lo largo del camino:

α=el valor de la mejor opción (es decir, valor más alto) que hemos encontrado hasta ahora en cualquier punto elegido a lo largo del camino para MAX.

β=el valor de la mejor opción (es decir, valor más bajo) que hemos encontrado hasta ahora en cualquier punto elegido a lo largo del camino para MIN.

La búsqueda alfa-beta actualiza el valor de α y β según se va recorriendo el árbol y poda las ramas restantes en un nodo (es decir, termina la llamada recurrente) tan pronto

como el valor del nodo actual es peor que el actual valor α o β para MAX o MIN, respectivamente.

En la Figura 2.6 se detalla los pasos del Algoritmo minimax con poda alfa-beta y en la Figura 2.5 un ejemplo de aplicación.

La eficacia de la poda alfa-beta es muy dependiente del orden en el que se examinan los sucesores. Por ejemplo, para un nodo del árbol se puede dar el caso de no realizar ninguna poda a sus descendientes inmediatos si se han generado primero los sucesores peores. Por el contrario si se genera primero el mejor sucesor, se podarían los restantes y el proceso de poda tendría mayor éxito.

Pues bien, la complejidad en tiempo, que antes era de $O(b^m)$ para minimax, se reduce ahora a $O(b^{m/2})$ aplicando poda alfa-beta. Esto significa que el factor de ramificación eficaz se transforma a $\sqrt{b}$ en vez de b (para el ajedrez, 6 en lugar de 35). Desde el punto de vista temporal, alfa-beta puede mirar hacia delante aproximadamente dos veces más que minimax en la misma cantidad del tiempo. Si los sucesores se examinan en el orden aleatorio más que primero el mejor, el número total de nodos examinados será aproximadamente $O(b^{3d/4})$ para un moderado b. Para el ajedrez, una función de ordenación bastante sencilla (como intentar primero las capturas, luego las amenazas, luego mover hacia delante, y, por último los movimientos hacia atrás) lo consiguen en aproximadamente un factor de dos del resultado $O(b^{m/2})$ del mejor caso. Añadir esquemas dinámicos que ordenan movimientos, como intentar primero los movimientos que fueron mejores la última vez, supone llegar bastante cerca del límite teórico.

Los estados repetidos en un árbol de búsqueda pueden causar un aumento exponencial del coste de búsqueda. En juegos, los estados repetidos ocurren con frecuencia debido a **transposiciones** (permutaciones diferentes de la secuencia de movimientos que acaba en la misma posición). Por ejemplo, en el ajedrez, si las blancas tienen un movimiento a_1 que puede ser contestado por las negras con b_1 y un movimiento no relacionado a_2 del otro lado del tablero puede ser contestado por b_2, entonces las secuencias $[a_1, b_1, a_2, b_2]$ y

```
función BÚSQUEDA-ALFA-BETA(estado) devuelve una acción
    variables de entrada:
        estado, estado actual del juego

    v ← MAX-VALOR(estado, −∞, +∞)
    devolver la acción de SUCESORES(estado) con valor v
```

```
función MAX-VALOR(estado, α, β) devuelve un valor utilidad
    variables de entrada:
        estado, estado actual del juego
        α, valor de la mejor alternativa para MAX a lo largo del camino a estado
        β, valor de la mejor alternativa para MIN a lo largo del camino a estado

    si TEST-TERMINAL(estado) entonces
        devolver UTILIDAD(estado)
    v ← −∞
    para a, s en SUCESORES(estado) hacer
        v ← MAX(v, MIN-VALOR(s, α, β)
        si v ≥ β entonces
            devolver v
        α ← MAX(α, v)
    devolver v
```

```
función MIN-VALOR(estado, α, β) devuelve un valor utilidad
    variables de entrada:
        estado, estado actual del juego
        α, valor de la mejor alternativa para MAX a lo largo del camino a estado
        β, valor de la mejor alternativa para MIN a lo largo del camino a estado

    si TEST-TERMINAL(estado) entonces
        devolver UTILIDAD(estado)
    v ← +∞
    para a, s en SUCESORES(estado) hacer
        v ← MIN(v, MAX-VALOR(s, α, β)
        si v ≤ α entonces
            devolver v
        β ← MIN(β, v)
    devolver v
```

Figura 2.6: Algoritmo minimax con poda alfa-beta

$[a_1, b_2, a_2, b_1]$ terminan en la misma posición (como las permutaciones que comienzan con a_2). Vale la pena almacenar la evaluación de esta posición en una tabla *hash* la primera vez que se encuentre, de modo que no tuviéramos que volver a calcularlo las siguientes veces. Tradicionalmente a la tabla *hash* de posiciones se le llama **tabla de transposición**. La utilización de una tabla de transposiciones puede tener un efecto espectacular a veces, tanto como doblar la profundidad accesible de búsqueda en el ajedrez. Por otra parte, si evaluamos un millón de nodos por segundo no es práctico guardar *todos* ellos en la tabla de transposiciones.

2.1.4. Refinamientos adicionales en la búsqueda

La efectividad del procedimiento alfa-beta depende en gran medida del orden en que se examinen los caminos. Si se examinan primero los peores caminos no se realizará ningún corte. Pero, naturalmente, si de antemano se conociera el mejor camino no necesitaríamos buscarlo.

La efectividad de la técnica de poda en el caso perfecto ofrece una cota superior del rendimiento en otras situaciones. Según Knuth y Moore (1975), si los **nodos** están perfectamente **ordenados**, el número de nodos terminales considerados en una búsqueda de profundidad d con alfa-beta es el doble de los nodos terminales considerados en una búsqueda de profundidad $\mathbf{d/2}$ sin alfa-beta. Esto significa que en el caso perfecto, una búsqueda alfa-beta nos da una ganancia sustancial pudiendo explorar con igual costo la mitad de los nodos finales de un árbol de juego del doble de profundidad.

Otra técnica para mejorar el rendimiento de la búsqueda se denomina **poda de inutilidades** y es la finalización de la exploración de un subárbol que ofrece pocas posibilidades de mejora sobre otros caminos que ya han sido explorados.

Puede implementarse también la técnica denominada **espera del reposo**. Cuando la condición de corte de la recursión del algoritmo minimax está basada sólo en la profundidad fija del árbol explorado, puede darse el llamado efecto horizonte (será explicado con más detalle en el punto 2.1.7). Esto ocurre cuando se evalúa como buena o mala

una posición, sin saber que en la siguiente jugada la situación se revierte. La espera del reposo ayuda a evitar el efecto horizonte. Una de las condiciones de corte de recursión en el algoritmo minimax debería ser el alcanzar una situación estable. Si se evalúa un nodo de un árbol de juego y este valor cambia drásticamente al evaluar el nodo después de realizar la exploración de un nivel más del árbol, la búsqueda debería continuar hasta que esto dejara de ocurrir. Esto se denomina esperar el reposo y nos asegura que las medidas a corto plazo, por ejemplo un intercambio de piezas, no influyen indebidamente en la elección.

Otra manera de mejorar el procedimiento de búsqueda es realizar una **búsqueda secundaria**, es decir, una doble comprobación del movimiento elegido, para asegurarnos que no hay una trampa algunos movimientos más allá de los que exploró la búsqueda inicial. Tras haber elegido un movimiento en concreto con un procedimiento minimax con alfa-beta que explora n niveles, la técnica de búsqueda secundaria realiza una búsqueda adicional de d niveles en la única rama escogida para asegurar que la elección sea buena.

La solución ideal de un juego y que nos devolvería soluciones inmediatas, sería seleccionar un movimiento consultando la situación actual del juego en una **base de datos de jugadas** y extrayendo el movimiento correcto. Naturalmente en juegos complicados esto es imposible de realizar, pues la tabla sería muy grande y además nadie sabría construirla. Sin embargo, este enfoque resulta razonable para algunas partes de ciertos juegos. Se puede usar movimientos de libro en aperturas y finales, combinando con el uso de un procedimiento de búsqueda para la parte central de la partida, mejorando el rendimiento del programa. Por ejemplo, las secuencias de apertura y los finales del ajedrez están altamente estudiados y pueden ser catalogados.

Una forma de podar aún más un árbol de juegos, es hacer que el factor de ramificación varíe a fin de dirigir más esfuerzo hacia las jugadas más prometedoras. Se puede realizar una **búsqueda sesgada** clasificando cada nodo hijo, quizás mediante un evaluador estático rápido y luego utilizando la siguiente fórmula:

$$R(hijo) = R(padre) - r(hijo)$$

donde R es el número de ramas de un nodo y r es el lugar que ocupa un nodo entre sus hermanos ordenados según la evaluación estática rápida.

2.1.5. Búsqueda por profundización iterativa

La profundización iterativa es una idea que se utilizó por primera vez en un programa llamado CHESS 4.5. El nombre profundización iterativa hace referencia a que se realizan iteraciones de búsquedas cada vez más profundas.

La principal razón para realizar iteraciones de búsquedas de diferentes profundidades incrementales, en lugar de realizar una búsqueda a la profundidad deseada, es que los programas de juegos pueden estar sujetos a restricciones de tiempo. Mediante esta técnica la primera iteración realiza una búsqueda de profundidad uno, la segunda iteración realiza una búsqueda de profundidad dos, etc. hasta que el tiempo destinado al movimiento se agote.

Puede parecer que se pierde una gran cantidad de tiempo al efectuar los análisis de los niveles menos profundos. Sin embargo, está demostrado que aproximadamente se necesita 1/(b-1) evaluaciones extras para realizar la exploración iterativa en un árbol de un factor de ramificación igual a b (por ej., si b=16 la profundización iterativa necesita 1/15 del total de exploraciones de los nodos hijos del último nivel).

Si se aplica el algoritmo minimax con poda alfa-beta descripto puede suceder que se encuentre fuera de tiempo cuando aún no ha recorrido todo el árbol y por lo tanto no tiene ninguna decisión tomada. Mediante la profundización iterativa puede cortarse la búsqueda en cualquier momento dado que siempre se tiene una solución resultado de la última iteración completada. Aclarar que en cada una de las iteraciones de la búsqueda no dejaríamos de utilizar el algoritmo minimax con poda alfa-beta.

Otra ventaja de la profundización iterativa es que cada iteración proporciona un valor para los movimientos, y por lo tanto esto puede proporcionar un orden. Es decir, si un movimiento es superior a sus hermanos puede explorarse primero en la siguiente iteración.

Con este ordenamiento, según ya se mencionó, el procedimiento de poda alfa-beta puede podar mayor cantidad de ramas y con esto reducirse el tiempo total.

2.1.6. Funciones de evaluación

Una función de evaluación devuelve una estimación de la utilidad esperada de una posición dada. La idea de una estimación no era nueva cuando Shannon la propuso. Durante siglos, los jugadores de ajedrez (y aficionados de otros juegos) han desarrollado modos de juzgar el valor de una posición, debido a que la gente es aún más limitada, en cantidad de la búsqueda, que los programas de computador. Debería estar claro que el funcionamiento de un programa de juegos es dependiente de la calidad de su función de evaluación. Una función de evaluación inexacta dirigirá a un árbol de búsqueda hacia posiciones que resultan estar perdidas. ¿Cómo se diseñan funciones de evaluación buenas?

Primero, la función de evaluación debería ordenar los estados *terminales* del mismo modo que la función de utilidad verdadera; por otra parte un agente que la use podrá seleccionar movimientos subóptimos aunque puede ver delante el final del juego. Segundo, ¡el cálculo no debe utilizar demasiado tiempo! (la función de evaluación podría llamar a la DECISIÓN MINIMAX como una subrutina y calcular el valor exacto de la posición, pero esto frustraría nuestro propósito de ahorrar tiempo). Tercero, para estados no terminales, la función de evaluación debería estar fuertemente correlacionalda con las posibilidades actuales de ganar.

Uno podría preguntarse sobre la frase *las posibilidades de ganar*. Después de todo, el ajedrez no es un juego de azar: sabemos el estado actual con certeza. Pero si la búsqueda debe cortarse en estados no terminales, entonces necesariamente el algoritmo será *incierto* sobre los resultados finales de esos estados. Este tipo de incertidumbre está inducida por limitaciones computaciones, más que de información. Si consideramos la cantidad limitada de cálculo que se permiten a la función de evaluación cuando se aplica a un estado, lo mejor que se pordría hacer es una conjetura sobre el resultado final.

Hagamos esta idea más concreta. La mayoría de las funciones de evaluación traba-

jan calculando varias características del estado (por ejemplo, en el juego de ajedrez, el número de peones capturados por cada lado). Las características, juntas, definen varias categorías o clases de equivalencia de estados: los estados en cada categoría dada, por lo general, tendrá algunos estados que conducen a triunfos, algunos que conducen a empates, y algunos que conducen a pérdidas. La función de evaluación no sabe cuál es cada estado, pero si puede devolver un valor que refleje la proporción de estados con cada resultado. Por ejemplo, supongamos que nuestra experiencia sugiere que el 72 por ciento de los estados encontrados en la categoría conduce a un triunfo (utilidad +1); el 20 por ciento a una pérdida (-1), y el 8 por ciento a un empate (0). Entonces una evaluación razonable de estados en la categoría es el valor medio ponderado o **valor esperado**: $0,72 \cdot (+1) + 0,20 \cdot (-1) + 0,08 \cdot 0 = 0,52$. En principio el valor esperado se puede determinar para cada categoría, produciendo una función de evaluación que trabaja para cualquier estado. Mientras que con los estados terminales la función de evaluación no tiene que devolver valores actuales esperados, la ordenación de los estados es el mismo.

En la práctica esta clase de análisis requiere demasiadas categorías y demasiada experiencia para estimar todas las probabilidades de ganar. En cambio la mayoría de las funciones de evaluación calculan las contribuciones numéricas de cada característica y luego las combinan para encontrar el valor total. Por ejemplo, los libros de ajedrez dan, de forma aproximada, el **valor material** para cada pieza: cada peón vale 1, un caballo o el alfil valen 3, una torre 5, y la dama vale 9. Otras características como la *estructura buena del peón* y *seguridad del rey* podrían valer la mitad de un peón, por ejemplo. Estos valores de características, entonces, simplemente se suman para obtener la evaluación de la posición. Matemáticamente, a esta clase de función de evaluación se le llama función ponderada lineal, porque puede expresarse como

$$\text{EVAL}(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s) = \sum_{i=1}^{n} w_i f_i(s)$$

donde cada w_i es un peso y cada f_i es una característica de la posición. Para el ajedrez los f_i podrían ser los números de cada clase de piezas sobre el tablero, y los w_i, podrían ser los valores de las piezas (1 para el peón, 3 para el alfil y caballo, 5 para la torre y 9

para la dama). En la Figura 2.7 se puede ver un tablero de ajedrez en la mitad de una partida. Veamos la evaluación de dicho tablero teniendo en cuenta que las negras puntuan positivo y las blancas negativo. Se ha tenido en cuenta únicamente el valor de las piezas indicadas anteriormente:

$$\text{EVAL}(s) = (1 \cdot 3 + 3 \cdot 2 + 5 \cdot 1 + 9 \cdot 1) - (1 \cdot 5 + 3 \cdot 1 + 5 \cdot 1) = 23 - 13 = 10$$

Observamos que las negras tienen una ventaja de 10 en material. Evidentemente la utilidad de un tablero al inicio de la partida es 0, ya que ambos jugadores comienzan con igualdad de condiciones.

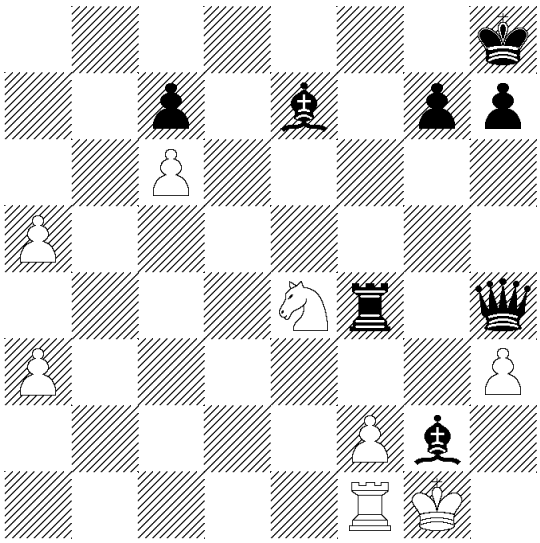

Figura 2.7: Tablero evaluado

La suma de los valores de las características parece razonable, pero de hecho implica un axioma muy fuerte: que la contribución de cada caracterísitca sea *independiente* de los valores de las otras características. Por ejemplo, la asignación del valor a un alfil no utiliza el hecho de que el alfil es más poderoso en la fase final, cuando tienen mucho espacio para maniobrar. Por esta razón, los programas actuales para el ajedrez, y otros juegos, también utilizan combinaciones *no lineales* de características. Por ejemplo, un par de alfiles podría merecer más la pena que dos veces el valor de un alfil, y un alfil merece más la pena en

la fase final que al principio.

Es de notar que las características y los pesos no son parte de las reglas del ajedrez. Provienen, desde hace siglos, de la experiencia humana al jugar al ajedrez. Considerando la forma lineal de evaluación, las características y los pesos dan como resultado la mejor aproximación a la ordenación exacta de los estados según su valor. En particular, la experiencia sugiere que una ventaja de más de un punto probablemente gane el juego, si no intervienen otros factores; una ventaja de 3 puntos es suficiente para una victoria. Uno de los objetivos de este proyecto será el aprendizaje óptimo de los pesos de las piezas junto con otros de posición de las mismas, pero esto se tratará en capítulos posteriores.

2.1.7. Corte de la búsqueda

El siguiente paso es modificar la búsqueda alfa-beta de modo que llame a la función heurística EVAL cuando se corte la búsqueda. También debemos llevar la contabilidad de la profundidad de modo que ésta se incremente sobre cada llamada recursiva. La aproximación más sencilla para controlar la cantidad de búsqueda es poner un límite de profundidad fijo, de modo que se corte la búsqueda para toda profundidad mayor que alguna profundidad fija d. La profundidad d se elige de modo que la cantidad de tiempo usado no exceda de lo que permiten las reglas del juego.

Una aproximación más robusta es aplicar profundidad iterativa. Cuando el tiempo se agota, el programa devuelve el movimiento seleccionado por la búqueda completa más profunda. Sin embargo, estas aproximaciones pueden conducir a errores debido a la naturaleza aproximada de la función de evaluación. Consdiere otra vez la función de evaluación simple para el ajedrez basada en la ventaja del material. Supongamos los programas de búsqueda al límite de profundidad y por ejemplo una situación de partida detectada por la búsqueda con muy favorable para las blancas, a un nivel más profundo y que no ha explorado por la búsqueda las negras provocan un jaque mate.

Obviamente se necesita un test del límite más sofisticado. La función de evaluación

debería aplicarse sólo a posiciones que son **estables** (es decir, improbablemente expuestas a grandes oscilaciones en su valor en un futuro próximo). En el ajedrez, por ejemplo, las posiciones en las cuales se pueden hacer capturas no son estables para una función de evaluación que solamente cuenta el material. Las posiciones no estables se pueden exstender hasta que se alcancen posiciones estables. A esta búsqueda suplementaria se le llama **búsqueda de estabilidad o de reposo**; a veces se restringe a sólo ciertos tipos de movimientos, como movimientos de captura, que resolverán rápidamente la incertidumbre en la posición.

El **efecto horizonte** es más difícil de detectar. Se produce cuando el programa afronta un movimiento del oponente, que causa un daño serio e inevitable. Considere el juego de ajedrez de la Figura 2.8. El negro aventaja en el material, pero si el blanco puede avanzar su peón de la séptima fila a la octava, el peón se convertirá en una dama y creará un triunfo fácil para el blanco. El negro puede prevenir este resultado temporalmente dando jaque con su torre, pero al final, inevitablemente el peón se convertirá en una reina. El problema con la búsqueda de profundidad fija es que se cree que esquivar estos movimientos evitan el movimiento de convertirse en una reina (inevitable) sobre el horizonte de búsqueda a un lugar donde no puede detectarse.

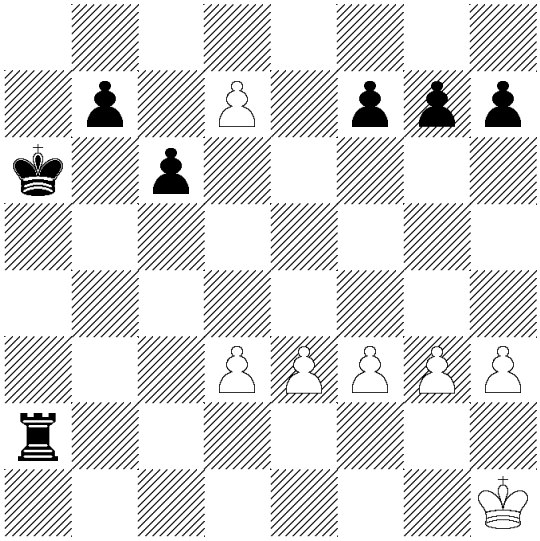

Figura 2.8: Situación de partida donde puede darse el efecto horizonte

Cuando las mejoras de hardware nos lleven a realizar búsquedas más profundas, se espera que el efecto horizonte ocurra con menos frecuencia (las secuencias que tardan mucho tiempo son raras). El uso de **extensiones excepcionales** también ha sido bastante eficaz para evitar el efecto horizonte sin añadir demasiado coste a la búsqueda. Una extensión excepcional es un movimiento que es *claramente mejor* que todos los demás en una posición dada. Una búsqueda de extensión excepcional puede ir más alla del límite de profundidad normal sin incurrir mucho en el coste porque su factor de ramificación es 1. (Se puede pensar que la búsqueda de estabilidad es una variante de extensiones excepcionales). En la Figura 2.8, una búsqueda de extensión excepcional encontrará el movimiento de convertirse en reina, a condición de que los movimientos de jaque de Negro y los movimientos del rey blanco puedan identificarse como *claramente mejores* que las alternativas.

Hasta ahora hemos hablado del corte de la búsqueda a un cierto nivel y que la poda alfa-beta, probablemente, no tiene nungún efecto sobre el resultado. Es también posible hacer la **poda hacia delante**, en la que podamos inmediatamente algunos movimientos de un nodo. Claramente, la mayoría de la gente que juega al ajedrez sólo considera unos pocos movimientos de cada posición (al menos conscientemente). Lamentablemente, esta aproximación es bastante peligrosa porque no hay ninguna garantía de que el mejor movimiento no sea podado. Esto puede ser desastroso aplicado cerca de la raíz, porque entonces a menudo el programa omitirá movimientos *evidentes*. La poda hacia delante puede usarse en situaciones especiales (por ejemplo, cuando dos movimientos son simétricos o equivalentes, sólo consideramos uno) o uno para nodos profundos en el árbol de búsqueda.

La combinación de todas las técnicas descritas proporciona un programa que puede jugar al ajedrez loablemente (o a otros juegos). Asumamos que hemos implementado una función de evaluación para el ajedrez, un test de límite razonable con una búsqueda de estabilidad, y una tabla de transposiciones grande. También asumamos que, después de meses de intentos tediosos, podemos generar y evaluar alrededor de un millón de nodos por segundo sobre los últimos PCs, permitiéndonos buscar aproximadamente 200 millones de nodos por movimiento bajo un control estándar de tiempo (tres minutos por movimiento). El factor de ramificación para el ajedrez es de 35, como promedio, y 35^5 son

aproximadamente 50 millones, así si usamos la búsqueda minimax podríamos mirar sólo 5 capas. Aunque no sea incompetente, tal programa puede ser engañado fácilmente por un jugador medio de ajedrez, el cual puede plantear, de vez en cuando, seis u ocho capas. Con la búsqueda alfa-beta nos ponemos en aproximadamente 10 capas, y resulta un nivel experto del juego. Para alcanzar el nivel de gran maestro necesitaríamos técnicas de poda adicionales que puedan ampliar la profundidad de búsqueda y una función de evaluación ajustada con una base de datos grande de movimientos de apertura y movimientos finales óptimos.

2.2. Modelos gráficos probabilísticos

Los modelos gráficos probabilísticos vienen a aunar las posibilidades de expresión de los grafos en procesos de razonamiento complejos, y toda la teoría de la probabilidad ampliamente integrada en el campo cíentífico.

Los modelos gráficos probabilísiticos representan distribuciones de probabilidad multivariantes. Esta reprentación se expresa mediante un productorio de términos, cada uno de los cuales abarca sólo unas pocas variables de todo el conjunto de variables consideradas. La estructura del producto se representa mediante un grafo que relaciona variables que aparecen en un mismo término.

Será el grafo el que especifique la forma del producto de las distribuciones y provea, asímismo, de herramientas para el razonamiento sobre propiedades vinculadas a los términos de dicho producto (Jensen, 1996). Para una grafo disperso, la representación será compacta y en muchos casos permitirá procesos eficientes de inferencia y aprendizaje.

Debido a su implantación dentro de la comunidad investigadora y a sus contrastados resultados, las redes bayesianas han sido el paradigma de modelo gráfico probabilístico seleccionado.

2.2.1. Redes Bayesianas

Sea un conjunto de variables aleatorias -atributos del problema- definido como

$$X = (X_1, ..., X_n)$$

La distribución de probabilidades conjuntas en una red Bayesiana se representa como un producto de probabilidades condicionadas. Cada variable aleatoria X_i tiene asociada una probabilidad condicional

$$p(X_i = x_i | PaX_i = pax_i)$$

donde $PaX_i \subset X$ es el conjunto de variables identificadas como padres de X_i. El producto resultado es de la forma:

$$p(X_1 = x_1, ..., X_n = x_n) = \prod_{i=1}^{n} p(X_i = x_i | PaX_i = pax_i)$$

La representación gráfica es un grafo dirigido en el que se establecen uniones dirigidas entre los padres de X_i y el propio X_i. Al tener en cuenta probabilidades condicionadas, el número de parámetros necesarios para determinar la distribución conjunta se reduce. En la Figura 2.9 y en Figura 2.10 podemos observar la reducción en ese número, pasando de 31 a 11 parámetros. La factorización conjunta para el modelo presentado en la Figura 2.9 y en Figura 2.10 se expresará como

$$p(x_1, x_2, x_3, x_4, x_5) = p(x_1) \cdot p(x_2|x_1) \cdot p(x_3|x_1) \cdot p(x_4|x_2, x_3) \cdot p(x_5|x_3)$$

Para obtener una Red Bayesiana es necesario definir sus dos componentes:

- **Estructura**: Es la parte *cualitativa* del modelo, especificándose la estructura mediante un grafo dirigido acíclico, o DAG, que refleje las dependencias condicionadas entre las variables (Figura 2.9). El concepto de independencia condicional entre tripletas de variables forma la base para entender e interpretar el campo de las redes Bayesianas.

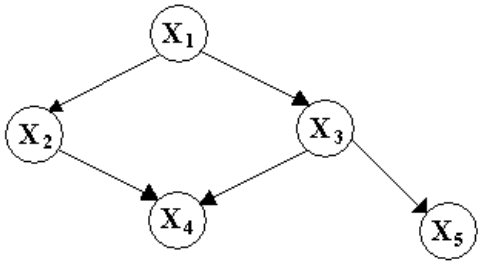

Figura 2.9: Estructura de una red Bayesiana

- **Parámetros**: Es la parte *cuantitativa* del modelo, y está formada por las probabilidades condicionadas y no condicionadas, o marginales, de las variables de la red (Figura 2.10). Aquellos nodos que no tengan padres, *nodos raíz*, tendrán asociadas una serie de probabilidades marginales. El resto de nodos de la red, o *nodos hijo*, tendrán asociadas unas probabilidades condicionadas en función de los valores que puedan tomar sus padres o antecesores.

$$
\begin{aligned}
&p(X_1 = 0) = 0,20\\
&p(X_2 = 0|X_1 = 0) = 0,80\\
&p(X_2 = 0|X_1 = 1) = 0,80\\
&p(X_3 = 0|X_1 = 0) = 0,20\\
&p(X_3 = 0|X_1 = 1) = 0,05\\
&p(X_4 = 0|X_2 = 0, X_3 = 0) = 0,80\\
&p(X_4 = 0|X_2 = 1, X_3 = 0) = 0,80\\
&p(X_4 = 0|X_2 = 0, X_3 = 1) = 0,80\\
&p(X_4 = 0|X_2 = 1, X_3 = 1) = 0,05\\
&p(X_5 = 0|X_3 = 0) = 0,80\\
&p(X_5 = 0|X_3 = 1) = 0,60
\end{aligned}
$$

Figura 2.10: Parámetros de la red Bayesiana

Una vez que la red Bayesiana ha sido construída, constituye un eficiente medio para realizar tareas de inferencia probabilista (Jensen, 1996). Dada una evidencia sobre el valor

de ciertas variables en la red, puede ser calculada la distribución de probabilidad sobre otra serie de variables que puedan ser de interés. Sin embargo, el problema de construir la red Bayesiana se mantiene.

Este problema puede ser abordado desde dos vertientes diferenciadas; una mediante la ayuda externa de expertos sobre el dominio a estudio -con el consiguiente gasto de tiempo y posibles errores en las aproximaciones-, o, dos, mediante el aprendizaje automático tanto de la estructura como de los parámetros en base a un conjunto de hechos conocidos sobre el dominio, es decir, sobre una base de datos de casos.

Por otro lado, la tarea de aprendizaje puede separarse en dos subtareas: la identificación de la topología de la red Bayesiana, esto es, el aprendizaje de la estructura; y el aprendizaje paramétrico como estimación de parámetros numéricos (probabilidades condicionadas y marginales) prefijada una topología de red Bayesiana.

Dentro del aprendizaje automático, existen dos grandes aproximaciones al problema de inducción de modelos Bayesianos: aproximaciones basadas en la detección de dependencias condicionadas entre tripletas de variables; y, métodos de *score+search*.

Los algoritmos que intentan averiguar la estructura de una red Bayesiana mediante la detección de dependencias condicionales tienen como entrada algunas relaciones de dependencia condicional entre algún subconjunto de variables del modelo; y, devuelven como salida un grafo dirigido acíclico, que representa un alto porcentaje de esas relaciones. Una vez que la estructura ha sido aprendida los parámetros necesarios son estimados a partir de una base de datos de casos. En (Castillo y otros, 1996) pueden consultarse más detalles sobre esta aproximación al aprendizaje de redes Bayesianas desde datos.

Un gran número de los algoritmos existentes en el campo de la modelización Bayesiana pertenecen a la categoría de métodos *score+search*. Para poder utilizar esta aproximación, es necesario definir una métrica que mida la bonanza de cada red Bayesiana candidata con respecto al fichero de casos. Además necesitamos también un procedimiento eficiente

que guíe el proceso de búsqueda a través de todos los posibles grafos dirigidos acíclicos. Tres de los *scores* más habituales son: la penalización de la verosimilitud máxima, *scores* Bayesianos conocidos como de verosimilitud marginal, y *scores* basados en la teoría de la información. Con respecto al procedimiento de búsqueda existen muchas alternativas diferentes en la literatura: búsqueda exhaustiva, enfriamiento estadístico, algoritmos genéticos, búsqueda tabú, etc. Para una revisión sobre los métodos de *score+search* en el aprendizaje de redes Bayesianas a partir de datos, puede ser consultado el artículo de (Heckerman et al., 1995).

En este proyecto se utilizará un aprendizaje paramétrico, esto es, de las tablas de probabilidad condicionadas y marginales de cada una de las variables de una red Bayesiana, cuya estructura se encuentra definida de antemano, ya que se trata de un clasificador. Más adelante se detallará dicho proceso de aprendizaje, pero antes veamos unas nociones básicas de clasificación.

2.3. Clasificación supervisada

Sea un conjunto de N hechos, cada uno caracterizado por $n+1$ variables. Las primeras n variables, $X_1, X_2, .., X_n$, serán denominadas *variables predictoras*, y la variable de índice $n+1$, identificada como C, será denominada *clase*. Estos datos pueden presentarse en forma de tabla (Cuadro 2.1), utilizando la siguiente notación:

1. x_i^j será el valor que en el j-ésimo hecho toma la variable predictora i-ésima, $i = 1, ..., n$ y $j = 1, ..., N$
2. c^j será la clase a la que pertenece el j-ésimo hecho.

Los hechos pueden ser denominados también *casos* o *instancias*; y, a las variables se les conoce también como *atributos* del problema supervisado.

El objetivo es conseguir un *modelo clasificatorio* que posibilite la predicción del valor de la variable C ante la llegada de un nuevo caso, formado por n variables predictoras

	X_1	X_2	...	X_i	...	X_n	C
1	x_1^1	x_2^1	...	x_i^1	...	x_n^1	c^1
⋮	⋮	⋮	⋱	⋮	⋱	⋮	⋮
j	x_1^j	x_2^j	...	x_i^j	...	x_n^j	c^j
⋮	⋮	⋮	⋱	⋮	⋱	⋮	⋮
N	x_1^N	x_2^N	...	x_i^N	...	x_n^N	c^N

Tabla 2.1: Elementos en una clasificación supervisada

y del que se desconoce el valor de C. Es decir, clasificar correctamente un nuevo hecho basándonos en las evidencias anteriores.

Para poder acometer ese objetivo de la mejor manera posible, es necesario disponer de una base de conocimiento amplia y de calidad. Amplia, ya que cuanto más número de casos se disponga sobre el problema mejores serán las aproximaciones obtenidas, y, de calidad, ya que la presencia de casos repetidos, ruido o errores en los datos, variables irrelevantes (que aporten poca información), o variables redundantes entre sí, puede hacer que el modelo no se ajuste correctamente a la realidad, siendo sus predicciones erróneas.

Es habitual que el número de variables predictoras sea elevado, o que no todas aporten la misma cantidad de información por lo que suelen aplicarse técnicas de *selección de variables* que ayuden a encontrar estas variables.

Las aplicaciones de este tipo de paradigmas clasificatorios son enormes ya que, en general, cualquier sistema cuyos resultados puedan ser agrupados en clases, y del que se disponga de una base de evidencias previas, es suceptible de ser analizado con esta técnica.

A la hora de resolver este tipo de problemas de clasificación supervisada, los campos de la estadística y el aprendizaje automático han desarrollado diferentes técnicas: Análisis discriminante, redes neuronales, clasificadores K-NN , sistemas clasificadores, árboles de clasificación, regresión logística, inducción de reglas y máquinas de soporte vectorial, entre otras. (Hernández y otros, 2004)

Los clasificadores Bayesianos son la apuesta realizada para resolver el problema supervisado en este texto. La teoría de redes Bayesianas nos proveé de la fundamentación matemática para abordar el problema clasificatorio.

2.3.1. Clasificadores bayesianos

Las variables consideradas en este tipo de clasificadores toman un número finito de posibles estados, *variables discretas*. Para el caso de variables que tomen valores en un rango infinito, *variables continuas*, el paradigma de los clasificadores Gaussianos puede resolver el problema.

Los clasificadores Bayesianos están basados en la formulación del Teorema de Bayes:

$$p(A|B) = \frac{p(A) \cdot P(B|A)}{P(B)}$$

donde:

1. $p(A)$ es conocido como la probabilidad *a priori* de que el suceso A sea cierto.
2. $p(A|B)$ es conocido como la probabilidad *a posteriori*, o, la probabilidad de que el suceso A sea cierto tras considerar B.
3. $p(B|A)$ es conocido como verosimilitud o *likelihood*, e indica la probabilidad de que el suceso B sea cierto, asumiendo que A lo es.
4. $p(B)$ es la probabilidad *a priori* de que el suceso B sea cierto. Actúa de coeficiente normalizador o estandarizador en la fracción.

Este teorema no sólo puede ser aplicado a sucesos, sino también a variables aleatorias, tanto unidimensionales como multidimensionales. Su formulación general es:

$$p(Y = y|X = x) = \frac{p(Y = y) \cdot p(X = x|Y = y)}{\sum_{i=1}^{r_y} p(Y = y_i) \cdot p(X = x|Y = y_i)}$$

Aplicando al problema de clasificación supervisada, tenemos que $Y = C$ es una variable unidimensional; mientras que $X = (X_1, X_2, \ldots, X_n)$ es una variable n-dimensional. X será la variable predictora, e Y la variable a predecir -la clase predicha por el modelo-.

Asumiendo una función de error 0/1, un clasificador Bayesiano $\gamma(x)$ asigna la clase con mayor probabilidad *a posteriori* dada una determinada instancia, es decir,

$$\gamma(x) = arg_c max\, p(c|x_1, \ldots, x_n)$$

donde c representa la variable clase, y $x_1, \ldots, x_n$ son los valores de las variables predictoras. Podemos expresar la probabilidad *a posteriori* de la clase de la siguiente manera:

$$p(c|x_1, \ldots, x_n) \propto p(c) \cdot p(x_1, \ldots, x_n|c)$$

Asumiendo diferentes factorizaciones para $p(x_1, \ldots, x_n|c)$ se puede obtener una jerarquía de modelos de creciente complejidad dentro de los clasificadores Bayesianos, hasta órdenes exponenciales de $2^{m \cdot n}$ siendo m y n el número de dimensiones de las dos variables aleatorias. En el presente proyecto ha sido considerado uno de los modelos más ampliamente utilizados por su sencillez y eficiencia en su computabilidad: Naïve Bayes.

2.3.2. Clasificador Naïve Bayes

Sin duda alguna se trata del modelo más simple de clasificación supervisada con redes bayesianas. En este caso, la estructura de la red es fija y sólo necesitamos aprender los parámetros (probabilidades). El fundamento principal del clasificador Naïve Bayes es la suposición de que todos los atributos son independientes conocido el valor de la variable clase. A pesar de que asumir esta suposición en el clasificador Naïve Bayes (NB) es sin duda bastante fuerte y poco realista en la mayoría de los casos, se trata de uno de los clasificadores más utilizados. Además diversos estudios demuestran que sus resultados son competitivos con otras técnicas (redes neuronales y árboles de decisión entre otras)

en muchos problemas y que incluso las superan en algunos otros. Como un ejemplo en el que el clasificador NB se está mostrando como una de las técnicas más eficaces, podemos citar la lucha contra el correo basura o *spam*. Muchos lectores de correo incorporan este clasificador para etiquetar el correo no solicitado.

La hipótesis de independencia asumida por el clasificador NB da lugar a un modelo gráfico probabilístico en el que existe un único nodo raíz (la clase), y en la que todos los atributos son nodos hoja que tienen como único padre a la variable clase. Se muestra la estructura del clasificador en la Figura 2.11.

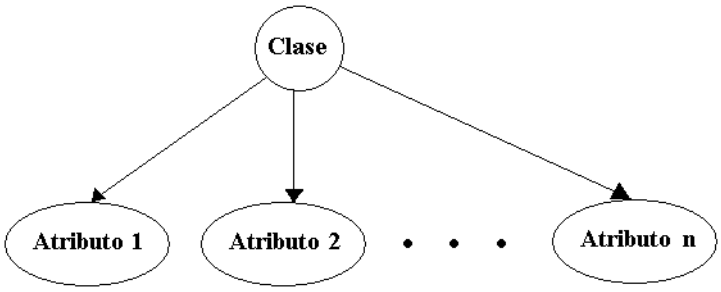

Figura 2.11: Topología de un clasificador Näive Bayes

2.3.2.1. Estimación de los parámetros

Debido a la hipótesis de independencia usada en el Naïve Bayes, la expresión para obtener la clase de máxima probabilidad a posteriori queda como sigue:

$$c_{MAP} = arg_{c \in \Omega_c} max\, p(A_1, \ldots, A_n|c)\, p(c) = arg\, max\, p(c) \prod_{i=1}^{n} p(A_i|c)$$

Es decir, la tabla de probabilidad $P(A_1, \ldots, A_n|c)$ ha sido factorizada como el producto de n tablas que sólo involucran dos variables. Por tanto, los parámetros que tenemos que estimar son $P(A_i|n)$ para cada atributo y la probabilidad *a priori* de la variable clase $P(c)$. Veamos cómo hacerlo dependiendo de que el atributo A_i sea discreto o continuo.

- **Atributos discretos**: en este caso la estimación de la probabilidad condicionada se basa en las frecuencias de aparición que obtendremos en la base de datos. Así, si

llamamos $n(x_i, Pa(x_i))$ al número de registros de la base de datos en que la variable X_i toma el valor x_i y los padres de $X_i(Pa(X_i))$ toman la configuración denotada por $Pa(x_i)$, entonces la forma más simple de estimar $P(x_i|Pa(x_i))$ es:

$$P(x_i|Pa(x_i)) = \frac{n(x_i, Pa(x_i))}{n(Pa(x_i))}$$

Es decir el número de casos favorables dividido por el número de casos totales. Esta técnica se conoce como *estimación por máxima verosimilitud* (*E.M.V.*) y tiene como desventaja que necesita una muestra de gran tamaño y que sobreajusta a los datos. Existen otros estimadores más complejos que palían estos problemas, entre ellos citaremos el *estimador basado en la ley de la sucesión de Laplace*:

$$P(x_i|Pa(x_i)) = \frac{n(x_i, Pa(x_i)) + 1}{n(Pa(x_i)) + |\Omega_{X_i}|}$$

Es decir, el número de casos favorables más uno dividido por el número de casos totales más el número de valores posibles. Nótese que con pocos ejemplos, la probabilidad se corrige por la probabilidad uniforme *a priori*, es decir, uno dividido por el número de valores posibles. Con esta estimación lo que se pretende, es que todas las configuraciones posibles tengan una mínima probabilidad, ya que con el estimador de máxima verosimilitud cualquier configuración que no esté presente en la base de datos tendría probabilidad 0. Por esta razón, el estimador utilizado en este proyecto a la hora de estimar las tablas de probabilidad del clasificador ha sido el *estimador basado en la ley de la sucesión de Laplace*.

- **Atributos continuos**: Aunque en este proyecto nuestros atributos van a ser discretos veamos cómo funciona el clasificador para el caso continuo. En este caso el clasificador Naïve Bayes supone que el atributo en cuestión sigue una distribución

normal dada la clase; por tanto, lo único que tenemos que calcular (a partir de la base de datos) es la media μ y la desviación típica σ condicionadas a cada valor de la variable clase.

$$P(A_i|c) \propto N(\mu, \sigma) = \frac{1}{\sqrt{2\pi} \cdot \sigma} exp\left[-\frac{(x-\mu)^2}{2\sigma^2}\right]$$

Evidentemente, esta estimación tiene el inconveniente de que los datos no siempre siguen una distribución normal.

Una vez que las tablas de probabilidad han sido estimadas ya se está en disposición de realizar una clasificación, esto es, la determinación de la clase más probable c_{MAP} se reduce a encontrar el conjunto de variables predictoras que maximizan la probabilidad de pertenencia a una determinada clase.

La gran **ventaja** del uso de este modelo es la sencillez en su formulación, y por tanto la de los cálculos a realizar. El tiempo de cómputo necesario para su aprendizaje y validación es **lineal** en el número de variables.

Pero existe un gran inconveniente, cuando la asunción de independencia condicional entre las variables predictoras no se cumple, la pérdida de precisión clasificatoria de este modelo puede ser grande. Este supuesto será violado cuando entre las variables predictoras existan redundancias, o bien, existan variables altamente correlacionadas unas con otras. Esto se ve compensado con la reducción del número de parámetros a estimar, lo que le hace especialmente útil en bases de datos pequeñas en relación con el número de variables, como es el caso de este proyecto.

2.3.2.2. Ejemplo de aplicación del clasificador Naïve Bayes

Supongamos que tenemos una pequeña base de datos de entrenamiento (Tabla 2.2), con datos acerca de los robos de coche registrados en un barrio marginal de Barcelona.

Caso	Color	Tipo	Origen	Robado?
1	Rojo	Deportivo	Propio	Si
2	Rojo	Deportivo	Propio	No
3	Rojo	Deportivo	Propio	Si
4	Amarillo	Deportivo	Propio	No
5	Amarillo	Deportivo	Alquilado	Si
6	Amarillo	Familiar	Alquilado	No
7	Amarillo	Familiar	Alquilado	Si
8	Amarillo	Familiar	Propio	No
9	Rojo	Familiar	Alquilado	No
10	Rojo	Deportivo	Alquilado	Si

Tabla 2.2: Base de datos de casos sobre robos de coches en un suburbio de Barcelona

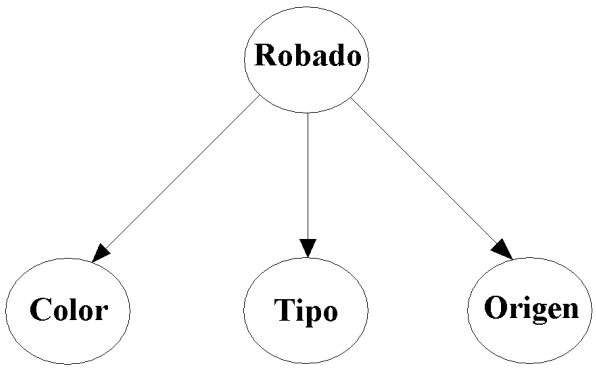

Figura 2.12: Estructura del clasificador de robos de coches

En la Figura 2.12 podemos observar la estructura del clasificador.

Los valores que pueden tomar las variables del clasificador son: *Robado = {Si, No}*; *Color = {Rojo, Amarillo}*; *Origen = {Propio, Alquilado}*

Vamos a calcular las tablas de probabilidad de las variables de la red usando para ello el estimador de la ley de sucesión de Laplace:

$$p(Si) = \frac{5+1}{10+2} = 0,5$$
$$p(No) = 0,5$$

$$p(Rojo|Si) = \frac{3+1}{5+2} \approx 0,57$$
$$p(Amarillo|Si) \approx 0,43$$

$$p(Rojo|No) = \frac{2+1}{5+2} \approx 0,43$$
$$p(Amarillo|No) \approx 0,57$$

$$p(Deportivo|Si) = \frac{4+1}{5+2} \approx 0,71$$
$$p(Familiar|Si) \approx 0,29$$

$$p(Deportivo|No) = \frac{2+1}{5+2} \approx 0,43$$
$$p(Familiar|No) \approx 0,57$$

$$p(Propio|Si) = \frac{2+1}{5+2} \approx 0,43$$
$$p(Alquilado|Si) \approx 0,57$$

$$p(Propio|No) = \frac{3+1}{5+2} \approx 0,57$$

$$p(Alquilado|No) \approx 0,43$$

Una vez tenemos el clasificador actualizado, imaginemos que queremos saber si un coche va a ser robado (al menos su probabilidad). Los datos de éste son: c= *{Rojo, Familiar, Propio}*. Lo que haremos será calcular la probabilidad *a posteriori* de que el coche sea robado condicionado a los atributos de c, y de que el coche no sea robado condicionado a los atributos de c. Tomaremos el máximo de estos dos valores y ya sabremos la probabilidad de que el coche sea robado.

$$p(Si|Rojo, Familiar, Propio) \propto p(Si) \cdot p(Rojo|Si) \cdot p(Familiar|Si) \cdot p(Propio|Si)$$

$$p(Si|Rojo, Familiar, Propio) \propto 0,5 \cdot 0,57 \cdot 0,29 \cdot 0,43 \approx 0,036$$

$$p(No|Rojo, Familiar, Propio) \propto p(No) \cdot p(Rojo|No) \cdot p(Familiar|No) \cdot p(Propio|No)$$

$$p(No|Rojo, Familiar, Propio) \propto 0,5 \cdot 0,43 \cdot 0,57 \cdot 0,57 \approx 0,070$$

Puesto que 0,070 >0,036, lo más probable es que el coche no sea robado.

Capítulo 3

Diseño del motor de juego de ajedrez

3.1. Punto de partida

Como ya es sabido, el objetivo del proyecto es el diseño de una red Bayesiana para el aprendizaje automático y para la adaptación del usuario en el juego del ajedrez. Por ello no he querido concentrar la mayor parte del esfuerzo en el diseño de una interfaz de juego con su validación de movimientos, etc. Busqué en la red algún código de libre distribución en Java que tuviera implementado por lo menos la interfaz del tablero (piezas y casillas) y la validación de los posibles movimientos (por ejemplo, el alfil mueve sólo en diagonal). Tras algunas pruebas con diversos códigos decidí tomar uno simple [MallardChess] que sólo tenía implementado una interfaz y validación de movimientos sencilla, para que a partir de ahí fuese relativamente fácil el posterior proceso de programación. En concreto, este código sólo da la posibilidad de jugar una partida un usuario contra otro usando el ratón y controlando los movimentos que son válidos o no, en cada momento. No tenía implementado nada acerca de la detección de jaque, jaque mate, cualquier tipo de tablas, y, por supuesto nada acerca de la *inteligencia* propia para jugar una partida con un usuario.

Así pues, mi punto de partida a la hora de realizar el ajedrez, comenzó en el diseño de todos estos aspectos que aún no estaban implementados y a partir de ahí diseñé el motor de juego de ajedrez que sería el que daría la *inteligencia* al juego. Veamos en los siguientes puntos cómo ha sido este proceso.

3.2. Definición de la heurística de juego inicial

3.2.1. Aspectos de rendimiento y calidad

Uno de los problemas que plantea la Inteligencia Artificial y en concreto la obtención de una solución óptima en problemas de búsqueda, es la complejidad de los algoritmos utilizados. En nuestro caso, la definición de la heurística de juego tiene un papel muy importante en la calidad de juego de un ajedrez por computador. Teniendo en cuenta que el algoritmo de búsqueda calcula la utilidad de cada nodo hoja del árbol de jugadas basándose en la heurística definida, ésta no debe ser muy costosa desde el punto de vista computacional. Por ello hay que buscar un compromiso entre una buena heurística y la eficiencia computacional.

En un principio, busqué heurísticas complejas que en su globalidad definían con mucho detalle los aspectos a tener en cuenta en un buen juego de ajedrez. Algunas de estos aspectos son:

- *Balance de material*, que engloba al número de piezas y la importancia de cada una de ellas
- *Centralización* de las piezas. Las 4 casillas centrales son las más importantes y por lo tanto deben ser premiadas
- Los *peones doblados* suponen una penalización, ya que pueden debilitar la defensa
- La *localización del peón* es más buena cuanto más adelantado está, ya que la coronación está más cerca. Por otro lado, los peones de los extremos tienen menos valor que los del centro
- Número de *piezas amenazadas* al contrario
- *Movilidad de las piezas*, que sería el número de posiciones a las que pueden mover nuestras piezas
- *Desarrollo de las piezas*, ya que aumentará el número de posiciones amenazadas al contrario

- *Torre en séptima fila*, ya que el rey contrario normalmente se encuentra aquí
- *Peones protegidos*, debido a su importancía sobre todo en los finales de juego
- *Peones pasados*, supone una buena situación, ya que el contrario tiene un peón menos para defender
- *Torres dobladas* sin piezas entre ellas, ya que se protegen una a otra
- Desarrollar el menor número de piezas antes que la dama, ya que ésta es la pieza con más movilidad
- *Enroque*, ya que aumenta la seguridad del rey
- *Mover el rey antes del enroque* se penaliza, ya que lo imposibilita
- *Jaque*, debido a que el contrario debe dedicarse a deshacerlo y no puede realizar cualquier jugada
- *Jaque Mate*, supone el fin de la partida
- *Seguridad del rey*, por ejemplo con piezas alrededor de él
- Conservación de la *pareja de alfil*, debido a que juntos pueden ser potentes
- *Torre en columna abierta*, ya que aumenta su movilidad
- Torre detrás de un peón con un desarrollo considerable, ya que lo protegemos para una posible coronación inmediata

Indicar que muchos de los aspectos anteriores se solapan y que su importancía varía según la situación de juego.

Pues bien, tras analizar algunos de los puntos anteriores, que en principio definen una heurística bastante buena, llegué a la conclusión de que la complejidad que suponía analizar tal cantidad de nodos con una heurística tan compleja era inabordable. En concreto analicé el número de piezas amenazadas al contrario, torre en séptima fila, peones doblados, centralización de peones, y alguna de menor importancia.

Por lo tanto, había que buscar una heurísita lo más sencilla posible y que englobara el mayor número de aspectos importantes definidos anteriormente, y además, por supuesto, que no fuese muy costosa computacionalmente. La heurística adoptada se detalla en el siguiente punto.

3.2.2. Bases de la heurística utilizada

A la hora de evaluar una posición del tablero de ajedrez junto con la historia de juego he partido de la siguiente heurística inicial que, por supuesto, no se trata de la heurística perfecta, de ahí el objetivo de aprendizaje del proyecto:

3.2.2.1. Material

A cada pieza sobre el tablero le asignamos un valor según su importancia. Los valores que se van a asignar y que coinciden en la mayoría de textos son:

- Peón: 100
- Caballo: 300
- Alfil: 300
- Torre: 500
- Dama: 900
- Rey: No se le asigna ningún valor debido a que no es posible su captura

3.2.2.2. Posicionamiento de las piezas

La posición de las piezas sobre el tablero es un aspecto muy importante en la heurística. Por tanto, se añadirá o restará importancia a una pieza según su localización sobre el tablero. Se utilizará para ello un array para cada pieza con 64 valores que ponderararán importancia de dicha pieza sobre el tablero. Desde la Figura 3.1 hasta la Figura 3.12 se detallan los pesos correspondientes a la posición de cada pieza. Estos valores han sido asignados de forma aproximada según el conocimiento de ajedrez de un principiante.

0	0	0	0	0	0	0	0
5	10	15	20	20	15	10	5
4	8	12	16	16	12	8	4
3	6	9	12	12	9	6	3
2	4	6	8	8	6	4	2
1	2	3	-10	-10	3	2	1
0	0	0	-40	-40	0	0	0
0	0	0	0	0	0	0	0

Figura 3.1: Pesos asociados a la posición del peón blanco

-10	-10	-10	-10	-10	-10	-10	-10
-10	0	0	0	0	0	0	-10
-10	0	5	5	5	5	0	-10
-10	0	5	10	10	5	0	-10
-10	0	5	10	10	5	0	-10
-10	0	5	5	5	5	0	-10
-10	0	0	0	0	0	0	-10
-10	-30	-10	-10	-10	-10	-30	-10

Figura 3.2: Pesos asociados a la posición del caballo blanco

-10	-10	-10	-10	-10	-10	-10	-10
-10	0	0	0	0	0	0	-10
-10	0	5	5	5	5	0	-10
-10	0	5	10	10	5	0	-10
-10	0	5	10	10	5	0	-10
-10	0	5	5	5	5	0	-10
-10	0	0	0	0	0	0	-10
-10	-10	-20	-10	-10	-20	-10	-10

Figura 3.3: Pesos asociados a la posición del alfil blanco

10	10	10	10	10	10	10	10
15	15	15	15	15	15	15	15
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	10	10	0	0	0
0	0	0	10	10	0	0	0
0	0	0	10	10	0	0	0

Figura 3.4: Pesos asociados a la posición de la torre blanca

10	10	10	10	10	10	10	10
15	15	15	15	15	15	15	15
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
5	5	0	0	0	0	5	5
5	5	0	0	0	0	5	5
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Figura 3.5: Pesos asociados a la posición de la dama blanca

-40	-40	-40	-40	-40	-40	-40	-40
-40	-40	-40	-40	-40	-40	-40	-40
-40	-40	-40	-40	-40	-40	-40	-40
-40	-40	-40	-40	-40	-40	-40	-40
-40	-40	-40	-40	-40	-40	-40	-40
-40	-40	-40	-40	-40	-40	-40	-40
-20	-20	-20	-20	-20	-20	-20	-20
0	20	40	-20	0	-20	40	20

Figura 3.6: Pesos asociados a la posición del rey blanco

0	0	0	0	0	0	0	0
0	0	0	-40	-40	0	0	0
1	2	3	-10	-10	3	2	1
2	4	6	8	8	6	4	2
3	6	9	12	12	9	6	3
4	8	12	16	16	12	8	4
5	10	15	20	20	15	10	5
0	0	0	0	0	0	0	0

Figura 3.7: Pesos asociados a la posición del peón negro

-10	-30	-10	-10	-10	-10	-30	-10
-10	0	0	0	0	0	0	-10
-10	0	5	5	5	5	0	-10
-10	0	5	10	10	5	0	-10
-10	0	5	10	10	5	0	-10
-10	0	5	5	5	5	0	-10
-10	0	0	0	0	0	0	-10
-10	-10	-10	-10	-10	-10	-10	-10

Figura 3.8: Pesos asociados a la posición del caballo negro

-10	-10	-20	-10	-10	-20	-10	-10
-10	0	0	0	0	0	0	-10
-10	0	5	5	5	5	0	-10
-10	0	5	10	10	5	0	-10
-10	0	5	10	10	5	0	-10
-10	0	5	5	5	5	0	-10
-10	0	0	0	0	0	0	-10
-10	-10	-10	-10	-10	-10	-10	-10

Figura 3.9: Pesos asociados a la posición del alfil negro

0	0	0	10	10	0	0	0
0	0	0	10	10	0	0	0
0	0	0	10	10	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
15	15	15	15	15	15	15	15
10	10	10	10	10	10	10	10

Figura 3.10: Pesos asociados a la posición de la torre negra

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
5	5	0	0	0	0	5	5
5	5	0	0	0	0	5	5
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
15	15	15	15	15	15	15	15
10	10	10	10	10	10	10	10

Figura 3.11: Pesos asociados a la posición de la dama negra

0	20	40	-20	0	-20	40	20
-20	-20	-20	-20	-20	-20	-20	-20
-40	-40	-40	-40	-40	-40	-40	-40
-40	-40	-40	-40	-40	-40	-40	-40
-40	-40	-40	-40	-40	-40	-40	-40
-40	-40	-40	-40	-40	-40	-40	-40
-40	-40	-40	-40	-40	-40	-40	-40
-40	-40	-40	-40	-40	-40	-40	-40

Figura 3.12: Pesos asociados a la posición del rey negro

3.2.2.3. Situación de Jaque

El tercer aspecto que tiene cuenta la función de evaluación es si existe una situación de jaque, en cuyo caso la función de evaluación lo premiará con un valor que inicialmente se ha asignado a 30.

3.3. Árbol de búsqueda

Tal y como anticipamos en el capítulo 2, el árbol de búsqueda utilizará el algoritmo minimax con una poda alfa-beta y una serie de comprobaciones para mejorar el rendimiento, esto es, cortes de búsqueda en situaciones de jaque mate o tablas por repetición o ahogo.

3.3.1. Problema encontrado y solución planteada

El procedimiento de búsqueda que utilizo es la profundización iterativa usando, en cada iteración de la misma, una búsqueda en profundidad. Las razones por las que hago esto se explican a continuación. La utilización de la poda alfa-beta provoca no encontrar la mejor solución, en nuestro caso el mejor movimiento, en el menor número de pasos posibles. Esto es debido a que este algoritmo busca la mejor solución mediante una búsqueda en profundidad. Por ejemplo, si a profundidad 4, se sabe con seguridad que una secuencia de movimientos nos llevará a un jaque mate y dicha secuencia se ha explorado antes que otra que nos da jaque mate a profundidad menor, la poda alfa-beta nos devolverá como mejor movimiento la primera opción, que aunque es buena no es la mejor. Si hay alguna situación crítica (jaque, tablas, ...) de partida a profundidad menor si será detectada con la solución que planteo. Esta razón y la limitación de tiempo que se puede especificar en el programa para que realice la búsqueda, me han llevado ha utilizar profundización iterativa.

En la Figura 3.13 se puede ver gráficamente el problema planteado. Observamos como una búsqueda en profundidad con límite 4 localizaría el jaque mate a esta profundidad y posteriormente podaría el resto de ramas conforme se va deshaciendo la recursividad del algoritmo. Por tanto no detectaría el jaque mate que hay a profundidad 2 en la

rama derecha. En la Figura 3.14 se puede observar el planteamiento de la búsqueda por profundización iterativa.

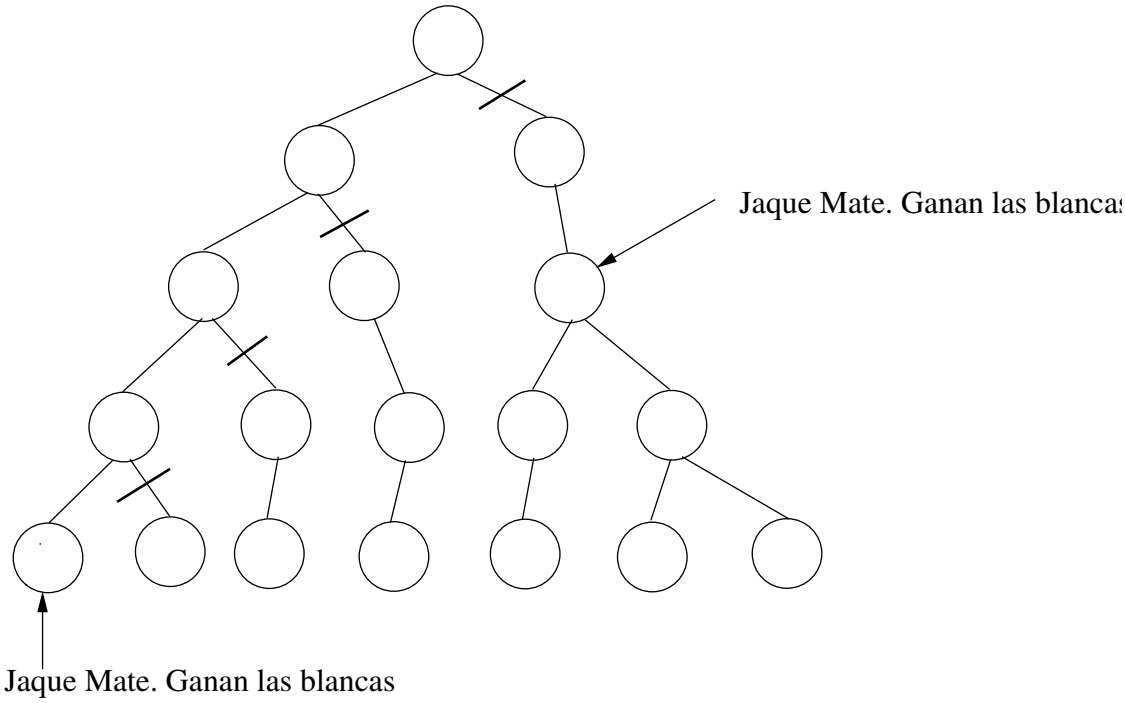

Figura 3.13: Problema de la búsqueda en profundidad en el juego del ajedrez

3.4. Localización de situaciones especiales

3.4.1. Jaque

Uno de los aspectos tenidos en cuenta a la hora del diseño del ajedrez es la localización de una situación de jaque. El jaque consiste en el ataque que se le hace directamente al rey por parte de alguna pieza adversaria. Para salirse de una posición de jaque, no hay otro sistema que capturar la pieza atacante, colocar una pieza para que obstruya el ataque o mover el rey a una casilla que no esté atacada. De lo que se deduce que el rey nunca puede ser capturado (de ahí que la heurística inicial no le asigne ningún valor). Sin embargo, un jaque producido por un caballo no se puede impedir ni obstruir colocando una pieza propia, puesto que, como ya es sabido, el caballo amenaza *saltando*. Por esta razón se dice

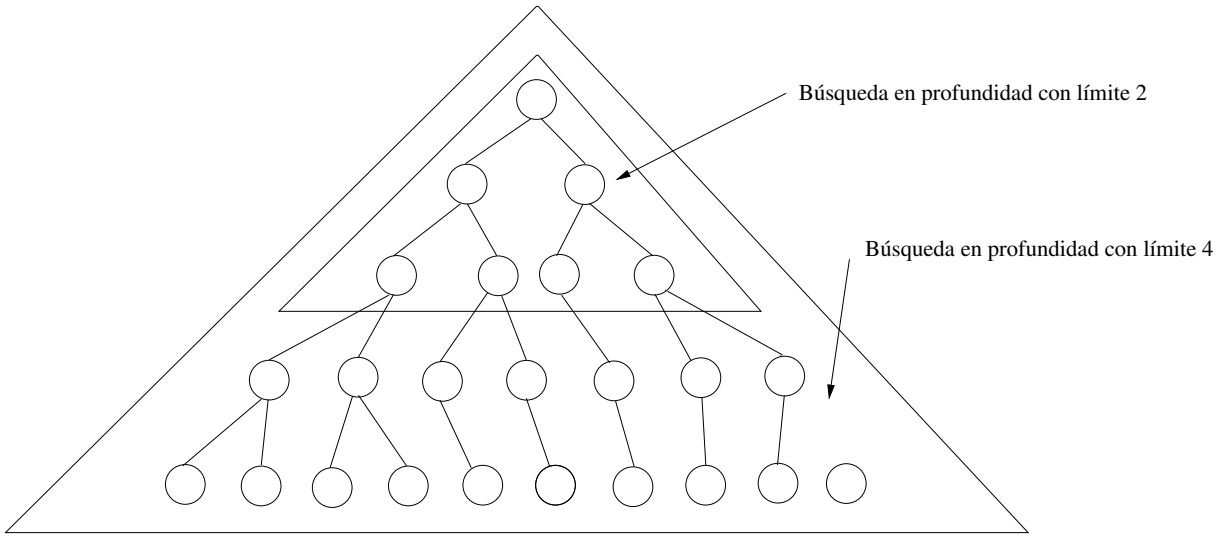

Figura 3.14: Búsqueda por profundización iterativa

que los jaques de caballo son siempre muy peligrosos.

Por otro lado, el rey no puede ponerse nunca en situación de jaque. Teniendo en cuenta esta regla, entenderemos que el rey nunca podrá capturar a una dama, aunque ésta se halle sin protección, porque si el rey no puede permanecer atacado, es incapaz de aproximarse a una dama, pieza que domina todas las casillas que la rodean. También le resultará imposible acercarse al rey contrario puesto que éste también domina todas las casillas inmediatas que le rodean.

En la Figura 3.15 se puede ver un ejemplo de jaque y en la Figura 3.16, un caso de limitación del movimiento del rey por amenaza del contrario. Se puede ver cómo el rey negro está en una casilla que generalmente tiene ocho movimientos, pero en esta posición el rey negro únicamente puede hacer tres movimientos. Le es imposible moverse a las casillas de la columna derecha, porque la torre ataca esas tres casillas y tampoco puede moverse en la propia columna que ocupa porque el caballo vulnera o amenaza dos casillas.

3.4.2. Fin de la partida

Otro aspecto a tener en cuenta ha sido determinar cuándo se llega al final de una partida debido a una situación de jaque mate, tablas por ahogo, tablas por repetición,

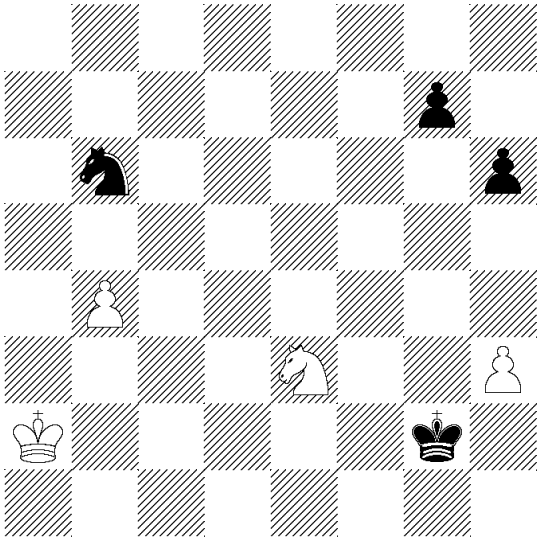

Figura 3.15: Ejemplo de jaque de caballo al rey negro

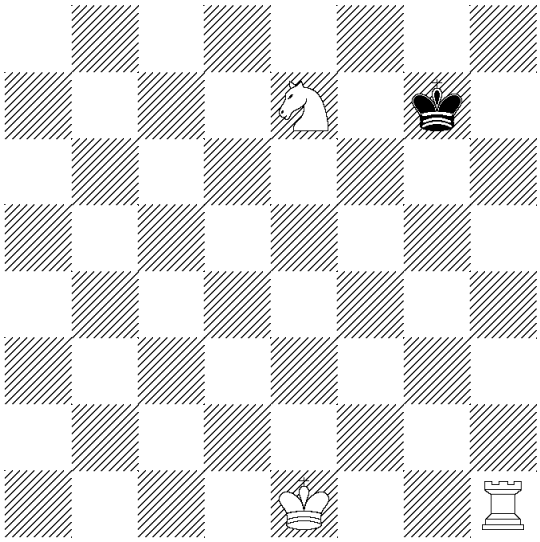

Figura 3.16: Limitación de movimiento del rey negro por amenaza blanca

tablas por la regla de las 50 jugadas o abandono. Veamos cada uno de ellos por separado.

3.4.2.1. Jaque Mate

Si un rey en jaque no puede moverse a ninguna casilla de las que le rodean porque están ocupadas por piezas propias, o por piezas contrarias defendidas, o bien porque están atacadas por piezas contrarias, y tampoco puede capturar la pieza que le amenaza o interponer otra pieza que obstruya el ataque, el rey está en posición de jaque mate. Se puede simplificar la definición diciendo que, cuando el rey no puede escapar o evitar el jaque porque no tiene casilla a donde ir, ni tampoco puede interrumpir la acción de las piezas contrarias, está en jaque mate. En la Figura 3.17 se puede observar un mate muy popular con los reyes enfrentados.

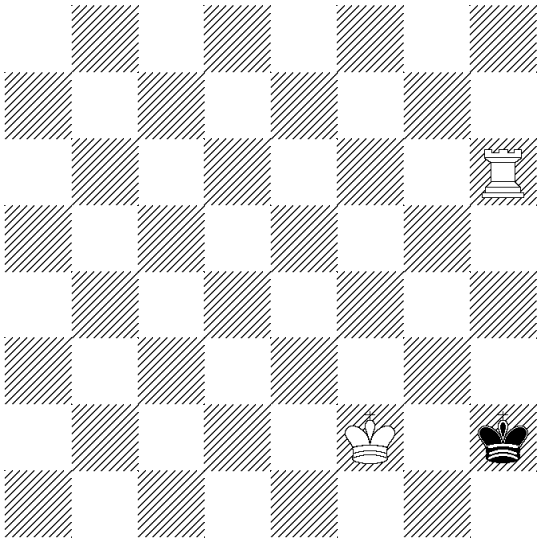

Figura 3.17: Situación de jaque mate

3.4.2.2. Tablas por ahogo

Cuando el jugador que debe mover queda imposibilitado para jugar sus piezas, y su rey no está en jaque, la partida entonces es tablas por ahogo. Esta situación suele producirse con relativa frecuencia. En la Figura 3.19 el turno es de las negras y se puede observar una situación de tablas por ahogo.

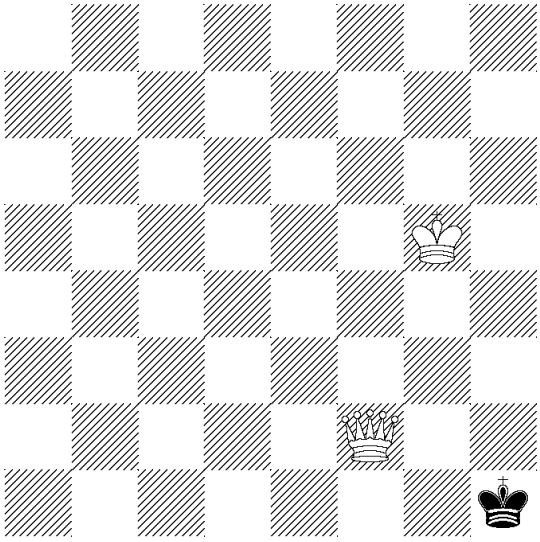

Figura 3.18: Situación de tablas por ahogo

3.4.2.3. Tablas por repetición

Cualquiera de los dos jugadores puede solicitar las tablas si se repite tres veces la misma posición durante la partida. Es absurdo que un jugador con ventaja en la partida provoque este tipo de tablas pero sí puede ser víctima de las mismas por el contrario. Puede llegarse a una posición en la que uno de los jugadores puede dar jaque al adversario repetidas veces sin que éste pueda evitarlo. A esto se le llama jaque perpetuo o continuo, y si esto sucede, la partida puede y debe declararse tablas. En la Figura 3.19 las blancas claramente tienen desventaja en la partida, sin embargo tienen una situación para poder provocar tablas por repetición. Si el turno es de las negras, el rey negro en jaque sólo puede mover a la casilla de la derecha. Si, posteriormente la dama blanca da de nuevo jaque moviéndose una casilla hacia la derecha, el rey negro deberá deshacer el jaque obligatoriamente moviendo de nuevo a la izquierda. Entonces si la dama blanca vuelve a mover a la izquierda y repite el mismo proceso tres veces, habrá conseguido tablas en una partida que tenía casi perdida.

3.4.2.4. Tablas por la regla de las 50 jugadas

Si en el transcurso de una partida de ajedrez se hacen 50 jugadas, por parte de ambos jugadores, y durante ellas no se ha llegado a mover ningún peón y no se ha capturado ninguna pieza, el juego debe considerarse empatado.

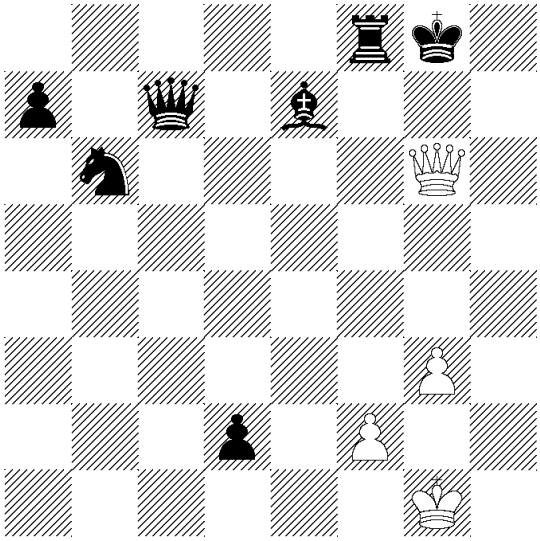

Figura 3.19: Situación de tablas por repetición

3.4.2.5. Tablas pactadas

Si un jugador decide que el empate sería un resultado justo y el adversario está de acuerdo, pueden firmarse las tablas; el juego puede considerarse empatado. También hay situaciones en las que se sabe que el material existente sobre el tablero no es suficiente como para decidir la partida; por ejemplo, rey contra rey, rey y caballo contra rey, etc. En estos casos también se firman las tablas.

3.4.2.6. Abandono

Raramente se llega a dar el mate en una partida de ajedrez, ya que el jugador que ve su partida por malos derroteros abandona. Por ello se ha incluido en la interfaz de la aplicación un botón que nos sirve para abandonar una partida que tenemos perdida. En el momento de pulsar este botón automáticamente se asignará la victoria al jugador contrario.

Capítulo 4

Aprendizaje automático

4.1. Fases de la partida

A la hora de generar los datos que forman parte de la base de datos de entrenamiento, se ha tenido en cuenta la fase en la que se encuentra la partida. Veámos cuáles son y los criterios seguidos para incluir una posición de tablero en una fase u otra:

- **Apertura**: Consideramos que una posición de tablero es de apertura, cuando ha sido generada durante los 10 primeros movimientos de juego. Un ejemplo de apertura se muestra en la Figura 4.1.

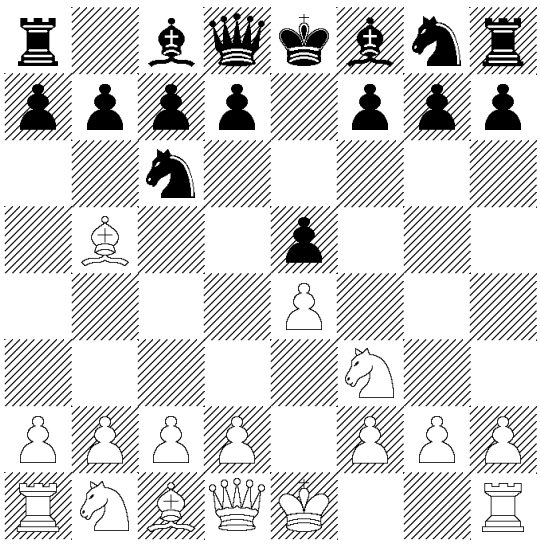

Figura 4.1: Ejemplo de partida en fase de apertura

- **Medio juego**: Una posición de tablero es considerada como de medio juego, cuando ha sido generada después de los primeros 10 movimientos de la partida, siempre con más de 10 piezas sobre el tablero y al menos una dama sobre el mismo. En la Figura 4.2 se observa una situación de medio juego.

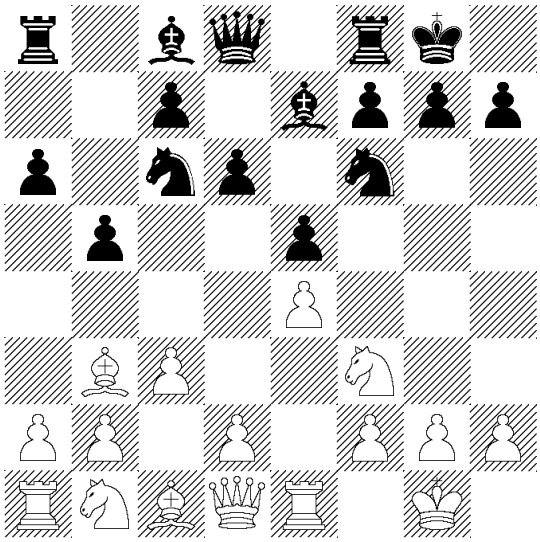

Figura 4.2: Ejemplo de partida en fase de medio juego

- **Final**: Una posición de tablero pertenece a la fase final, cuando quedan 10 o menos piezas o no quedan damas sobre el tablero. En la Figura 4.3 y 4.4 se pueden observar dos ejemplos de partida en fase final.

4.2. Resultados posibles en una partida

Aclaremos explícitamente cuáles son los posibles resultados de una partida de ajedrez, ya que, posteriormente, ésta será una variable clase del clasificador junto con la fase de la partida. Tomando como referencia a la máquina tenemos tres:

- **Ganar**: La máquina realizar un jaque mate sobre el oponente o éste abandona.
- **Perder**: La máquina es vencida con jaque mate o abandona.
- **Tablas**: Se produce una situación de tablas por ahogo, repetición, pactadas, ..., quedando la partida empatada.

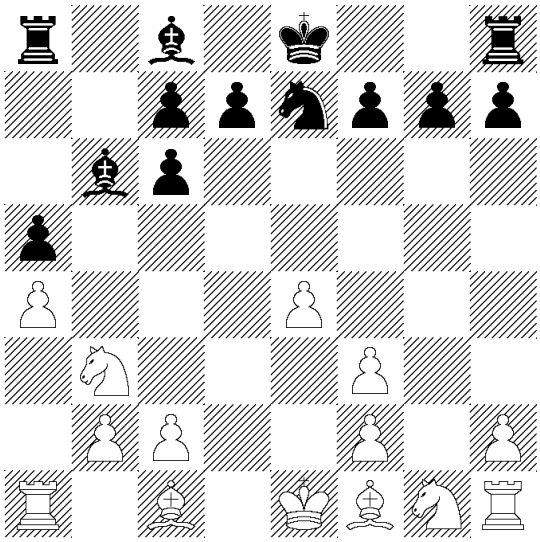

Figura 4.3: Ejemplo de partida en fase final sin damas sobre el tablero

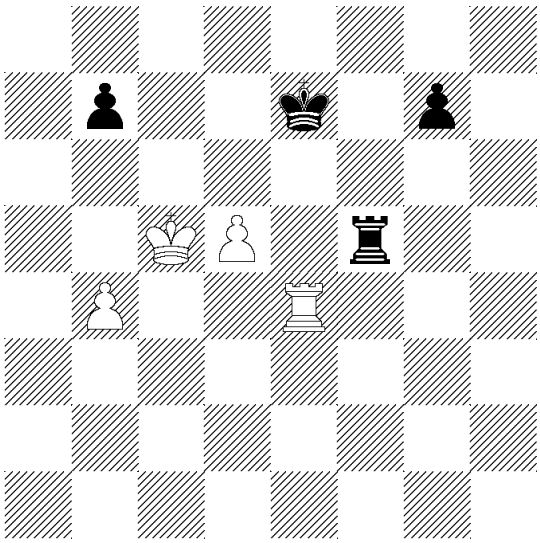

Figura 4.4: Ejemplo de partida en fase final con un número de piezas inferior a diez

4.3. Construcción del clasificador para el aprendizaje

El objetivo de este clasificador será obtener, con ayuda de una base de datos de entrenamiento con partidas, la configuración de parámetros, definidos en la heurística inicial, óptima en resultado y fase, para seguir una estrategia en concreto. Por ejemplo, podemos recoger del clasificador aquella configuración de valores de las variables que en fase de apertura nos permite aumentar la probabilidad de ganar, o por ejemplo aquéllos que nos minimicen la probabilidad de perder en la fase media de juego. También podríamos obtener, por ejemplo, aquéllos que nos permitan aumentar o disminuir la probabilidad de hacer tablas en la fase final de juego, ya que en un momento determinado esto puede ser interesante.

Los parámetros de los que hablamos serán las variables del clasificador, junto con la fase de la partida y el resultado de la misma.

4.3.1. Variabilidad de los parámetros de la heurística

A la hora de generar una base de datos de entrenamiento se ha procedido a enfrentar a la máquina con ella misma, usando un color la heurística inicial, definida por nosotros, y otro una heurística variada de forma aleatoria según una serie de criterios que a continuación se detallan.

Para cada parámetro de la heurística se obtiene un valor entre cinco posibles. La forma de obtener estos valores difiere si se trata del valor de las piezas, posiciones o jaque:

- **Variabilidad del valor de las piezas**: Tomando como referencia el valor inicial v, obtendremos los cuatro valores restantes $v1, v2, v3, v4$, sumando y restando a éste el 20 % y 40 % de su valor. Veámoslo:

$$v1 = v - |v| \cdot 0,4$$

$$v2 = v - |v| \cdot 0,2$$

$$v3 = v + |v| \cdot 0,2$$

$$v4 = v + |v| \cdot 0,4$$

- **Variabilidad del valor posicional de las piezas**: Tomando como referencia el valor inicial v, obtendremos, en este caso, los cuatro valores restantes $v1, v2, v3, v4$, sumando y restando a éste el 100 % y 200 % de su valor. Veámoslo:

$$v1 = v - |v| \cdot 2$$

$$v2 = v - |v| \cdot 1$$

$$v3 = v + |v| \cdot 1$$

$$v4 = v + |v| \cdot 2$$

  Puesto que existen algunos valores de posición que valen 0, y por tanto no podríamos generar variaciones, ya que todas nos darían 0, he optado por sumar y restar los valores 3 y 6 al original. De esta forma permito la variabilidad de estos valores, ya que de lo contrario estaríamos perdiendo información. Por tanto para variables con valor 0, los cuatros valores restantes $v1, v2, v3, v4$ son:

$$v1 = -6$$

$$v2 = -3$$

$$v3 = 3$$

$$v4 = 6$$

- **Variabilidad del valor de jaque**: La variabilidad asignada a este valor v es igual al valor de las piezas. Veámoslo:

$$v1 = v - |v| \cdot 0,4$$

$$v2 = v - |v| \cdot 0,2$$

$$v3 = v + |v| \cdot 0,2$$

$$v4 = v + |v| \cdot 0,4$$

La razón por la que he seleccionado estos porcentajes de variabilidad para cada tipo de parámetros está justificada. He aplicado una menor variabilidad al valor de las piezas y el jaque, debido a que la asignación inicial seleccionada es la que haría la mayoría de expertos en ajedrez, por tanto conviene no rotarlos mucho puesto que la solución óptima se encuentra muy probablemente cerca del valor inicial. Sin embargo el valor posicional de las piezas es algo más *aprendible*, esto es, los valores inicialmente asignados seguramente no sean los más óptimos y por tanto conviene rotarlos a fin de obtener aproximaciones mejores.

4.3.2. Variables del clasificador

1. **Fase de la partida**: Ésta será la primera de las dos variables clase del clasificador y nos indicará en qué fase de la partida se encuentra una situación de tablero determinada. Será una variable observada, ya que en cada momento sabremos con certeza en la fase de la partida en la que nos encontramos. Los valores que puede tomar esta variable son:

$$Fase\ de\ la\ partida = \{Apertura,\ Medio,\ Final\}$$

2. **Resultado**: Ésta será la otra variable clase del clasificador y nos especificará cuál ha sido el resultado de la partida. También será una variable observada, lo determinaremos para establecer la estrategia de juego, como veremos más adelante. Los valores posibles de esta variables son:

$$Resultado = \{Ganar,\ Perder,\ Tablas\}$$

3. **Peón**: Ésta es la primera de las variables que se van a aprender. Su valor inicial es de 100. Indica la importancia del peón en el juego y sus posibles valores son:

$$Valor\ Peon = \{60, 80, 100, 120, 140\}$$

4. **Caballo**: Esta variable indica la importancia del caballo durante el juego. Su valor inicial es de 300 y los valores posibles que pueden tomar son:

$$Valor\ Caballo = \{180, 240, 300, 360, 480\}$$

5. **Alfil**: Indica la importancia del alfil en el juego. Se valora inicialmente a 300, igual que el caballo, así que los posibles valores son idénticos:

$$Valor\ Alfil = \{180, 240, 300, 360, 480\}$$

6. **Torre**: Indica la importancia de la torre en el juego. El valor inicial para esta variable es de 500 y sus posibles valores son:

$$Valor\ Torre = \{300, 400, 500, 600, 700\}$$

7. **Dama**: Esta el la pieza más importante del tablero, valorada con 900. Los valores que puede tomar son:

$$Valor\ Dama = \{540, 720, 900, 1080, 1260\}$$

8. **Jaque**: Esta variable indica la importancia del valor del jaque durante el juego. El valor está inicialmente asignado es 30 y el resto de posibles valores son:

$$Valor\ Jaque = \{18, 24, 30, 36, 42\}$$

9. **Peón en a8**: Este valor indica la importancia de un peón blanco en la casilla a8 del tablero. El valor inicialmente asignado es 0 y sus posibles valores son:

$$Peon\ en\ a8 = \{-6, -3, 0, 3, 6\}$$

De la misma forma se obtendrían cada uno de los 63 valores posicionales restantes del peón blanco. Y el mismo proceso con todas las piezas restantes diferenciadas por color. Para el caso del rey se dispone además de otro tablero que nos indica la importancia de su situación en finales de partida. En total este clasificador contiene 840 variables, incluyendo las dos variables clase.

4.3.3. Estructura del clasificador

En la Figura 4.5 se puede observar la estructura del clasificador utilizado para el aprendizaje automático. Como se puede apreciar se tienen dos variables clase que son las que observaremos y en función de ellas obtendremos una serie de parámetros que formarán la configuración deseada en cada momento. Los puntos suspensivos sustituyen al resto de variables posicionales de las piezas especificadas con anterioridad.

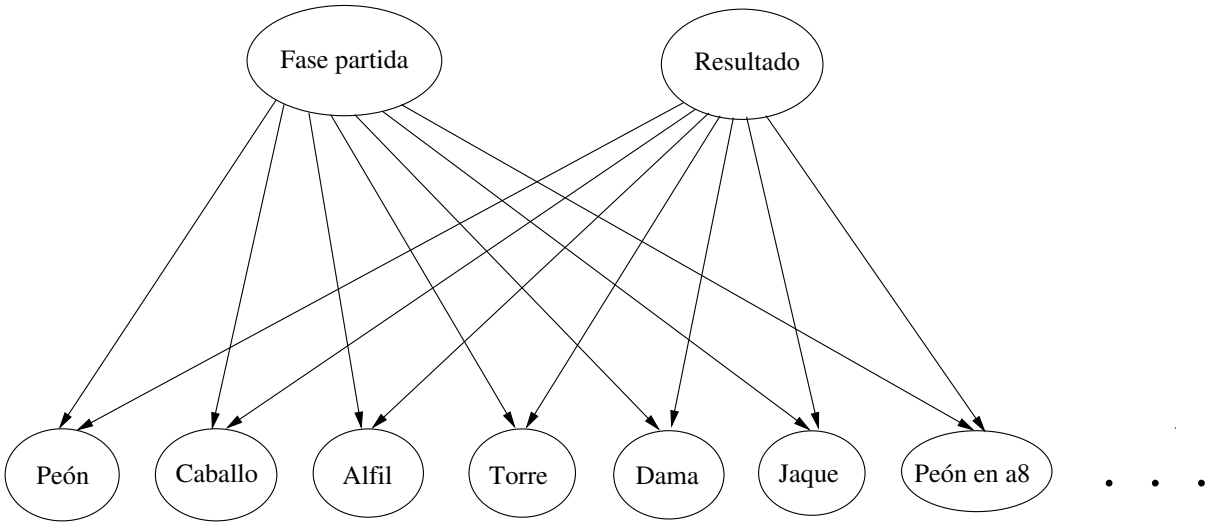

Figura 4.5: Estructura del clasificador para el aprendizaje automático

4.3.4. Generación de la base de datos de entrenamiento

Una vez que tenemos la estructura del clasificador clara tenemos que completar las tablas de probabilidad de las variables, exceptuando las de las dos variables clase, ya que en nuestro problema en concreto se trata de dos variables observadas. Para ello será necesario disponer de una base de datos de entrenamiento, en este caso de partidas, para lo cual se procederá a dejar un tiempo la máquina jugando con ella misma, usando en un bando una heurística fija y en otro una heurística aleatoria teniendo en cuenta la variabilidad comentada anteriormente. Para hacer la muestra más homogénea se dispondrá de la posibilidad de asignar a las blancas la heurística fija y a las negras la aleatoria, y vicerversa. Las partidas que el usuario juegue contra la máquina también serán almacenadas. En la Figura 4.6 se puede observar la ventana de la aplicación para generar una base de datos

de partidas.

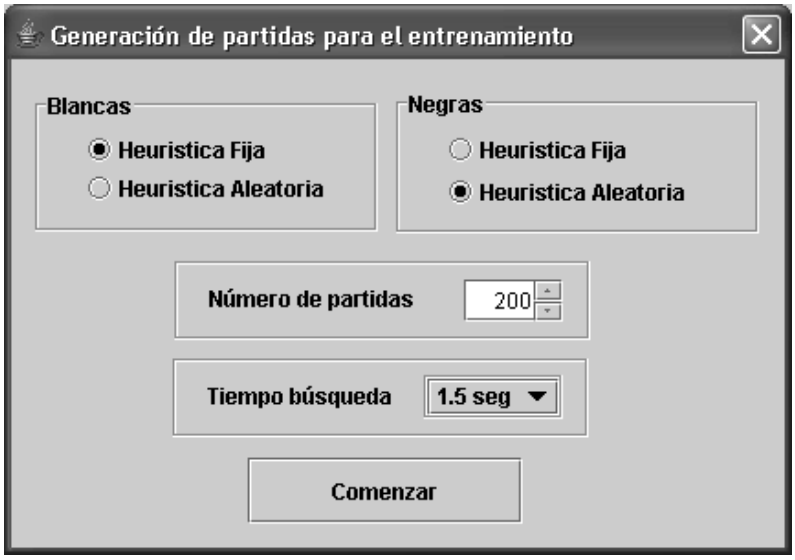

Figura 4.6: Generación de la base de datos de partidas

En la Tabla 4.1 se puede ver el formato de dicha base de datos con unos cuantos casos de ejemplo. Vemos cómo para cada etapa de cada partida se ha generado una configuración de parámetros aleatorios que serán los que utilizará la heurística en la etapa concreta de la partida en juego. Al final de cada caso aparece el resultado de la misma. Evidentemente el resultado en cada *caso* de la base de datos perteneciente a una misma partida es el mismo, de ahí que aparezca repetido de forma consecutiva.

Fase partida	Peon	Caballo	Alfil	Torre	Dama	Jaque	Peón a8	Peon b8	...	Resultado
APERTURA	120	180	240	700	540	42	-6	0	...	PIERDEN
MEDIO	120	360	360	600	1260	36	3	0	...	PIERDEN
FINAL	120	420	180	400	720	42	-3	3	...	PIERDEN
APERTURA	140	360	420	300	1080	18	0	6	...	GANAN
MEDIO	100	300	360	500	1260	42	3	0	...	GANAN
FINAL	80	180	240	400	900	42	6	6	...	GANAN
APERTURA	120	420	420	400	720	18	-6	3	...	TABLAS
MEDIO	140	300	360	500	1260	42	3	0	...	TABLAS
FINAL	120	420	180	600	900	30	0	6	...	TABLAS

Tabla 4.1: Ejemplo de casos de la base de datos de partidas

4.3.5. Formato de las tablas de probabilidad

A partir de la base de datos de partida previamente creada, se generarán las tablas de probabilidad de las variables usando para ello el estimador basado en la ley de sucesión de Laplace, explicado en el punto 2.3.2.1.

Los valores de las tablas de probabilidad del clasificador dependenderán del número de casos de la base de datos de partidas por lo que su valor no es fijo. En el siguiente capítulo veremos como los valores de las tablas de probabilidad del clasificador de estrategia del oponente son fijos ya que la base de datos de entrenamiento se establecerá en un principio y posteriormente no se modificará.

En el clasificador actual, la tabla de probabilidad tendrá 45 valores, ya que para cada uno de los 5 posibles valores de una variable habrá que diferenciar la fase de la partida y el resultado conseguido. En la Tabla 4.2 podemos observar un ejemplo de la tabla de probabilidad de la variable *Peón*. Se ha usado la abreviatura A, M y F para las fases de partida y G, P y T para el resultado.

	FASE PARTIDA	A	A	A	M	M	M	F	F	F
	RESULTADO	G	P	T	G	P	T	G	P	T
PEÓN	60	0,2	0,1	0,3	0,3	0,2	0,3	0,2	0,3	0,2
	80	0,3	0,1	0,1	0,1	0,2	0,1	0,2	0,1	0,2
	100	0,1	0,1	0,2	0,4	0,2	0,1	0,1	0,1	0,2
	120	0,1	0,2	0,4	0,1	0,3	0,1	0,3	0,2	0,3
	140	0,3	0,5	0,1	0,2	0,1	0,4	0,2	0,3	0,1

Tabla 4.2: Ejemplo de tabla de probabilidad de la variable *Peón*

4.4. Proceso de aprendizaje

El proceso de aprendizaje del juego no tiene límite, ya que dependerá del número de partidas existente en la base de datos de partidas. Por tanto, cuantas más partidas se hayan jugado más se refinarán los parámetros de la heurística, y por tanto mejor será el juego de la máquina.

Cada vez que el programa arranca se actualiza el clasificador para tener en cuenta las partidas jugadas y que se encuentran almacenadas en la base de datos. Puede indicarse explícitamente que se actualice el clasificador en una misma ejecución del programa, pero casi no tiene sentido ya que unas cuantas partidas jugadas en comparación con la base de datos, que debe ser grande para que el clasificador tenga éxito, no suponen mucho cambio.

Por tanto, una vez que tenemos el clasificador actualizado, el juego podrá usarlo para obtener la configuración de parámetros óptima que nos interese en cada momento. ¿Cuál es esta configuración óptima? Aquí entra en juego el clasificador explicado en el siguiente capítulo. Veamos unas pinceladas ahora.

Cuando la estrategia del oponente sea atacante, el clasificador deberá obtener la configuración de parámetros que minimice la probabilidad de perder; en cambio, cuando ésta sea defensiva, el clasificador obtendrá la configuración de parámetros que maximice la probabilidad de ganar. Para el caso mixto se seleccionará aleatoriamente entre una de las dos opciones anteriores.

Pero, ¿cómo se obtiene, por ejemplo, la configuración de parámetros que maximiza la probabilidad de que se dé un resultado? Debido a la propiedad de **independencia** de las variables en un clasificador Naïve Bayes, la configuración de parámetros que maximiza la probabilidad de un resultado, es igual al producto de las probabilidades de el parámetro de cada una de las variables que maximiza la probabilidad para una fase de partida y resultado concreto. De la misma forma, tan sólo cambiando maximiza por minimiza se obtendría la configuración de parámetros que minimiza la probabilidad de que se de un resultado en una etapa en concreto.

Por ejemplo, para el caso del peón podemos observar en la Tabla 4.3, señalado en negrita, cuál es el valor que maximiza la probabilidad de **ganar** en la fase **media** del juego. Buscaríamos la columna perteneciente a medio juego cuyo resultado es ganar, y obtendríamos el parámetro que maximiza la probabilidad. El valor en este caso es 100.

	FASE PARTIDA RESULTADO	A G	A P	A T	**M** **G**	M P	M T	F G	F P	F T
	60	0,2	0,1	0,3	0,3	0,2	0,3	0,2	0,3	0,2
	80	0,3	0,1	0,1	0,1	0,2	0,1	0,2	0,1	0,2
PEÓN	**100**	0,1	0,1	0,2	**0,4**	0,2	0,1	0,1	0,1	0,2
	120	0,1	0,2	0,4	0,1	0,3	0,1	0,3	0,2	0,3
	140	0,3	0,5	0,1	0,2	0,1	0,4	0,2	0,3	0,1

Tabla 4.3: Ejemplo de selección del valor de *Peón* más probable para ganar en la fase media del juego

Capítulo 5

Adaptación a la situación del oponente

En el capítulo anterior se ha explicado cómo una heurística de juego mediocre puede aprender a partir de una base de datos de entrenamiento. Lo que se pretende aquí es que dicha heurística aprendida se adapte, además, en cada momento, a la estrategia de juego del usuario al que se enfrenta. De esta forma la heurística no juega siempre igual. Pero, ¿qué nos aporta el reconocimiento de la estrategía de juego del contrincante? Pues bien, algo muy importante, y es que en el mundo ajedrecístico la mejor respuesta a un jugador con perfil atacante, es otro jugador con perfil defensivo, es decir, que concentre todos los esfuerzos en no perder. En cambio, si un jugador es defensivo, la mejor respuesta es un jugador que juegue al ataque, esto es, que concentre todos sus esfuerzos en ganar (no es lo mismo jugar a ganar que jugar a no perder, y ninguna de ellas tiene porqué ser una mala estrategia). De ahí que sea importante que la máquina reconozca el tipo de usuario que tiene enfrente para jugarle de una forma u otra, según le convenga.

5.1. Tipos de estrategias de juego

Los tipos de estrategía considerados para la situación del oponente han sido tres:

- **Atacante**: Esta estrategia la puso de moda en su época Robert Fischer, con su juego al ataque desde el inicio de las partidas. No le fue mal ya que llegó a ser un gran maestro del ajedrez. Otro jugador, retirado hace escasos días y considerado atacante es Gary Kasparov, hombre que ha dominado el ajedrez desde los 80. El juego de éste no es tan agresivo como el de Robert Fischer. En la Figura 5.1 se puede observar una situación de partida atacante en donde Fischer juega con blancas.

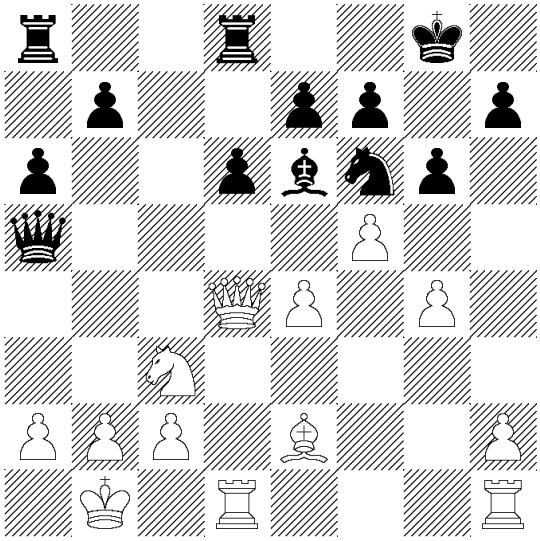

Figura 5.1: Situación de partida atacante con blancas (primer movimiento: e4)

- **Defensiva**: El jugador por excelencia que siguió esta estrategia es Anatoli Karpov, un jugador muy posicional, que prepara muy bien las jugadas y que juega más a la expectativa que los jugadores atacantes. En la Figura 5.2 se puede observar una situación de partida de Karpov jugando con negras. Se ve claramente el estilo posicional del jugador.

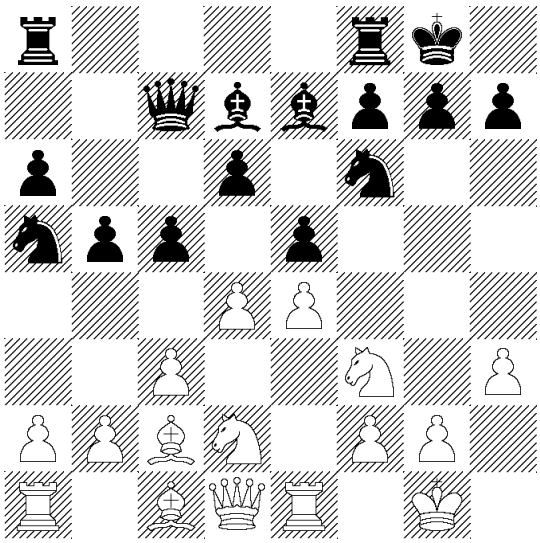

Figura 5.2: Situación de partida defensiva con negras (primer movimiento: e5)

- **Mixta**: Por último se ha considerado una estrategia mixta para clasificar aquéllos jugadores con un juego intermedio entre atacante y defensivo. Uno de los jugadores

que se aproxima más a este tipo de juego es Miguel Illescas, un español, cuyo tipo de juego no está totalmente definido, no queriendo decir con esto que sea malo ni mucho menos. En la Figura 5.3 se puede observar una situación de partida de Illescas jugando con blancas. Observamos como su primer movimiento Cf3, su enroque igual al contrario y una situación de las piezas normal lo caracterizan como un jugador más *imprevisible* y, por tanto, mixto.

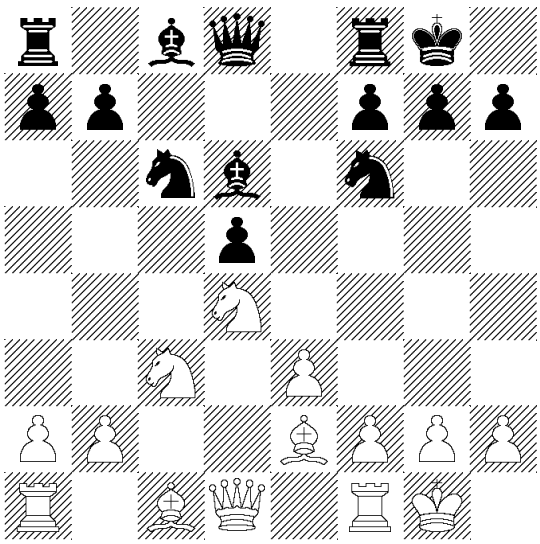

Figura 5.3: Situación de partida mixta con blancas (primer movimiento: Cf3)

5.2. Construcción del clasificador para la adaptación

Vamos a construir un clasificador Naïve Bayes para obtener la estrategia de juego de un usuario tomando como referencia una serie de características que serán las variables de dicho clasificador.

5.2.1. Variables

Las variables seleccionadas para que nos clasifiquen a un jugador como *Atacante*, *Defensivo* o *Mixto* son:

1. **Situación del oponente**: Ésta es la variable clase del clasificador y será la que nos determinará la estrategia de un jugador en función de los valores de las variables que se definen posteriormente. Los valores que puede tomar esta variable son:

Situación del oponente = {Atacante, Defensivo, Mixto}

2. **Primer movimiento**: Se ha seleccionado esta variable, ya que muchos jugadores de ajedrez, cuya estrategia es bien conocida, se identifican más con un primer movimiento que con otro, así que esto nos puede valer para clasificar jugadores. Por tanto, se almacenará el primer movimiento del jugador al que queremos caracterizar su estrategia. Notar que si éste juega con negras habrá que esperar al segundo movimiento de la partida, en caso contrario, tomaremos el primero, ya que jugaría con blancas. Esta variable puede tomar los siguientes valores:

   Primer movimiento = {c4, d4, e4, f4, Cf3, otro_b, c5, d5, e5, f5, Cf6, otro_n}

   en donde cada uno de ellos es un movimiento en notación algebraica (vease apéndice).

3. **Enroque**: Se tendrá en cuenta también la situación de los reyes. Si ambos se localizan en distintos flancos significará probablemente que el jugador que haya provocado esta situación esté jugando al ataque. En caso contrario (ambos reyes en el mismo flanco), es probable que se esté usando una estrategia más defensiva. Para el que no esté familiarizado con el ajedrez, los flancos, se componen de cuatro columnas del tablero cada uno. Está el flanco de dama (cuatro primeras columnas del tablero) y el flanco de rey (últimas cuatro columnas del tablero). Veamos qué valores puede tomar esta variable:

   Enroque = {Iguales, Opuestos}

4. **Número de piezas, exceptuando peones, por encima de la tercera fila**: Esta variable es más intuitiva que las anteriores. Es normal pensar que cuantas más piezas adelantadas tengamos más agresivo será nuestro juego. Los valores definidos para esta variable son:

   Numero de piezas adelantadas = {0, 1, 2, 3, >3}

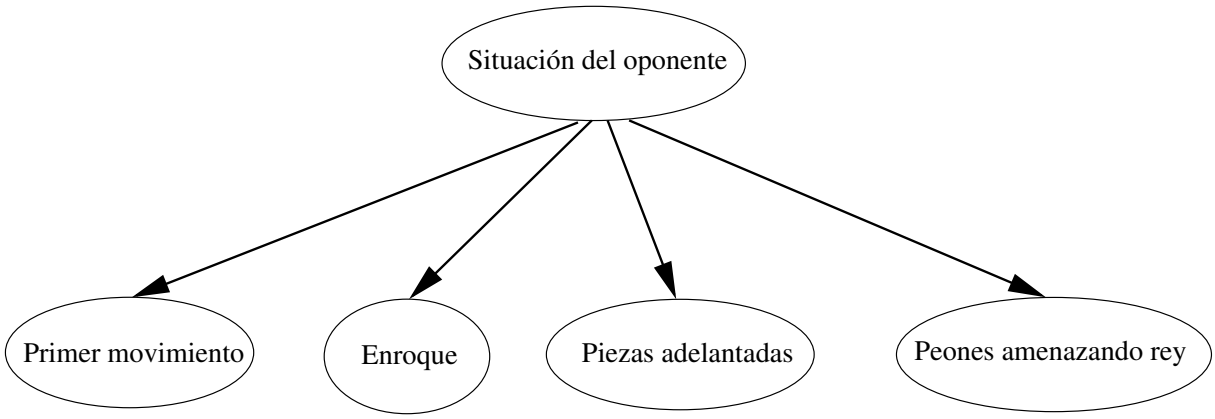

Figura 5.4: Estructura del clasificador de la estrategia del oponente

5. **Número de peones en la columna del rey contrario y sus adyacentes situados por encima de la tercera fila**: Esta variable completa a la anterior y no se ha tenido en cuenta ahí debido a la importantcia que tienen los peones en su conjunto. Por tanto cuanto mayor sea el número de peones que tengamos en la situación descrita mayor será nuestra agresividad en el juego. Los valores definidos para esta variable son:

$$\textit{Numero peones amenazando rey} = \{0,\ 1,\ 2,\ >2\}$$

5.2.2. Estructura del clasificador

En la Figura 5.4 se puede observar la estructura del clasificador usado para obtener la estrategia de juego de un usuario.

5.2.3. Generación de la base de datos de entrenamiento

Una vez tenemos la estructura del clasificador debemos calcular los parámetros, es decir, las tablas de probabilidad de todas las variables, exceptuando la de la clase, ya que no la necesitamos, al ser en nuestro problema en concreto una variable observada. Para ello, generaremos una base de datos con valores de las variables que constituyen el clasificador. Lo que haremos será tomar una colección de partidas de jugadores que por excelencia se consideran atacantes, defensivos y mixtos. En concreto, hemos seleccionado 2708 partidas de Robert Fischer y Gary Kasparov como modelo de juego atacante, 3078 de Anatoli Karpov como modelo de juego defensivo y, por último, 649 de Miguel Illescas

como modelo de juego mixto. Dichas partidas se encuentran en notación algebraica en su forma estándar (véase apéndice).

A partir de esta colección, se ha observado para cada tablero distinto de cada una de las partidas el valor de las 4 variables del clasificador. A estos 4 valores se le asigna el tipo de jugador (atacante, defensivo o mixto) de la partida analizada constituyendo, de esta forma, un nuevo caso para la base de datos de entrenamiento del clasificador de la estrategia del oponente. El número de casos generados para la base de datos es igual a todas las situaciones de tablero obtenidas de todas las partidas y aproximadamente son unos 265.000.

En la Tabla 5.1 se puede ver el formato de dicha base de datos con unos cuanto casos de ejemplo.

1^{er} movimiento	Enroque	Piezas adelantadas	Peones amenazando rey	Estrategia
e4	iguales	2	1	Atacante
Cf6	opuestos	0	2	Atacante
Cf3	iguales	0	1	Mixta
d4	iguales	1	0	Defensiva
c5	iguales	0	0	Atacante
c4	opuestos	1	2	Defensiva
otro_n	iguales	1	1	Mixta

Tabla 5.1: Ejemplo de casos de la base de datos de entrenamiento del clasificador de estrategias

5.2.4. Tablas de probabilidad

A partir de la base de datos previamente creada, se generarán las tablas de probabilidad de las variables usando para ello el estimador basado en la ley de sucesión de Laplace, explicado en el punto 2.3.2.1. Desde la Figura 5.5 hasta la Figura 5.8 se muestran las tablas de probabilidad obtenidas en formato de porcentajes, para una mayor claridad.

1er movimiento	ATACANTE	DEFENSIVO	MIXTO
c4	4,66%	5,92%	6,50%
d4	17,92%	19,12%	9,07%
e4	28,40%	25,97%	19,83%
f4	0,04%	0,02%	0,00%
Cf3	3,04%	4,08%	15,73%
otro_b	0,37%	0,14%	0,00%
c5	15,67%	5,70%	18,37%
d5	3,41%	3,26%	14,24%
e5	1,17%	7,88%	2,28%
f5	0,00%	0,12%	0,75%
Cf6	21,41%	21,14%	10,60%
otro_n	3,91%	6,66%	2,62%

Figura 5.5: Tabla de probabilidad de la variable *primer movimiento*

Enroque	ATACANTE	DEFENSIVO	MIXTO
Iguales	91,06%	93,51%	92,90%
Opuestos	8,94%	6,49%	7,10%

Figura 5.6: Tabla de probabilidad de la variable *enroque*

Nº piezas > 3ª fila	ATACANTE	DEFENSIVO	MIXTO
0	45,06%	47,70%	47,89%
1	35,96%	36,13%	35,32%
2	15,30%	13,46%	13,41%
3	3,38%	2,53%	3,06%
>3	0,30%	0,19%	0,31%

Figura 5.7: Tabla de probabilidad de la variable *piezas por encima de la 3ª fila*

Nº peones alrededor rey	ATACANTE	DEFENSIVO	MIXTO
0	54,98%	60,18%	59,68%
1	36,02%	32,59%	32,45%
2	8,17%	6,72%	7,47%
>2	0,83%	0,50%	0,40%

Figura 5.8: Tabla de probabilidad de la variable *peones amenazando al rey contrario*

5.3. Reconocimiento de la estrategia del oponente

Una vez que el clasificador está construído y actualizadas sus tablas de probabilidad con la base de datos de entrenamiento, ya estamos en condiciones de reconocer el tipo de juego en un instante determinado del usuario. El proceso consistirá en clasificar una configuración de valores que toman las 4 variables que disponemos, por ejemplo, supongamos que en un momento de la partida la situación del oponente está caracterizada con la siguiente configuración:

$$C = \{e4, iguales, 2, 1\}$$

esto es, juega con blancas siendo su primer movimiento *e4*, su rey está situado en el mismo flanco que el contrario, dispone de dos piezas distintas de peones por encima de la tercera fila y un peón por encima de la tercera fila en la columna del rey contrario o adyacentes. Pues bien, el clasificador realizará las siguientes operaciones:

- En primer lugar calculará tres probabilidades:

$$\textbf{P(Atacante|e4, iguales, 2, 1)} \propto P(e4|Atacante) \cdot P(iguales|Atacante) \cdot P(2|Atacante) \cdot P(1|Atacante) \propto 0,2840 \cdot 0,9106 \cdot 0,1530 \cdot 0,3602 \approx \textbf{0,01425}$$

$$\textbf{P(Defensivo|e4, iguales, 2, 1)} \propto P(e4|Defensivo) \cdot P(iguales|Defensivo) \cdot P(2|Defensivo) \cdot P(1|Defensivo) \propto 0,2597 \cdot 0,9351 \cdot 0,1346 \cdot 0,3259 \approx \textbf{0,01065}$$

$$\textbf{P(Mixto|e4, iguales, 2, 1)} \propto P(e4|Mixto) \cdot P(iguales|Mixto) \cdot P(2|Mixto) \cdot P(1|Mixto) \propto 0,1983 \cdot 0,9290 \cdot 0,1341 \cdot 0,3245 \approx \textbf{0,00802}$$

- Por último, selecciona la estrategia cuya probabilidad condicionada a los valores de la configuración es máxima. Puesto que $max(0,01425, 0,01065, 0,00802) = 0,01425$, la estrategia seleccionada es **atacante**.

Hacer notar que, en el cálculo de las tres probabilidades anteriores, no se ha tenido en cuenta el factor $P(Atacante)$, $P(Defensivo)$ y $P(Mixto)$, respectivamente, tal y como se especifica en el teorema de Bayes. La razón es que el número de partidas seleccionadas de

cada tipo de estrategia, no ha sido algo aleatorio, sino que ha dependido de una decisión propia. Si no tuviéramos en cuenta esto, estaríamos premiando a la estrategía cuyo número de partidas en la base de datos fuera mayor y estaríamos cometiendo un error. Por tanto, hemos considerado equiprobables a priori las tres posibles estrategias del oponente.

5.4. Proceso de adaptación al oponente

Una vez que sabemos el tipo de juego que está desarrollando nuestro oponente debemos adaptarnos a él. Aquí es donde ambos clasificadores - el primero diseñado para el aprendizaje de los parámetros de juego y el segundo el que tratamos en este capítulo - se relacionan. Veamos cómo lo hacen:

- Cuando la estrategia del oponente sea **atacante**, obtendremos del clasificador de aprendizaje aquellos parámetros que **minimizan la probabilidad de perder**. Es decir, si el usuario está jugando al ataque buscaremos una configuración de parámetros que sea defensiva.

- Cuando la estrategia del oponente sea **defensiva**, obtendremos del clasificador de aprendizaje aquellos parámetros que **maximizan la probabilidad de ganar**. Es decir, si el usuario está jugando defensivo buscaremos una configuración de parámetros que sea atacante.

- Cuando la estrategia del oponente no se encuentra definida en ninguna de las anteriores se seleccionará **aleatoriamente** del clasificador de aprendizaje una de las dos opciones tomadas anteriormente.

- Los tres puntos anteriores se aplican en la fase de apertura y medio juego, ya que en finales no tiene mucho sentido intentar seguir una estrategia de juego, lo importante es ganar, o en su caso no perder. Por tanto he aprovechado la etapa final para aplicar otro tipo de adaptación. Cuando la máquina tiene una desventaja material considerable (entorno a 200 puntos en la heurística aplicada), ésta seleccionará del clasificador de aprendizaje aquella configuración de parámetros que **maximice la probabilidad de hacer tablas** ya que, en sus condiciones, sería un resultado muy

bueno. En cambio, si es ella la que posee la ventaja, intentará no hacer tablas, esto es, **minimizar la probabilidad de hacer tablas**.

Capítulo 6

Resultados y conclusiones.

El proceso de aprendizaje de los parámetros de la heurística de juego puede observarse de forma clara con los resultados que se van a mostrar en este capítulo.

Disponemos de una base de datos de 200 partidas generadas de la siguiente forma:

- 100 partidas, en donde las blancas jugaban con la heurística inicial planteada y las negras con una heurística aleatoria basada en esta inicial.
- 100 partidas, igual pero al contrario, en este caso las blancas jugaban con la heurística aleatoria y las negras con la inicial.

A partir de estos datos, vamos a exponer dos experimentos que demuestran la eficacia del clasificador en el proceso de aprendizaje.

6.1. Torneo entre la heurística aprendida y la aleatoria

Lo que se pretende es demostrar que la heurística aprendida, que se obtiene del clasificador de aprendizaje, va *mejorando*. Pero, ¿cómo se puede saber de forma objetiva que realmente mejora?

Pues bien, lo que haremos será enfrentar a la heurística aleatoria con la heurística aprendida, y notaremos que cuantas más partidas tenga la base de datos más fuerte será la heurística aprendida frente a la aleatoria. Es decir, en un torneo de partidas la heurística de juego aprendida ganará más partidas a la aleatoria, cuantos más casos haya en la base

Resultado	**0**	**20**	**40**	**60**	**80**	**100**	**120**	**140**	**160**	**180**	**200**
GANA	7	5	6	6	7	7	8	10	10	9	10
PIERDE	6	6	5	6	4	3	1	3	2	2	2
TABLAS	2	4	4	3	4	5	6	2	3	4	3

Tabla 6.1: Torneo entre heurística aprendida frente a aleatoria

de datos. Incrementaremos el número de partidas disponibles, a la vez que enfrentamos a ambas heurísticas.

En la Tabla 6.1 se puede ver el resultado del experimento. Para cada tamaño de la base de datos múltiplo de 20 (desde 0 a 200 partidas), se jugará un torneo de 15 partidas entre la heurística aprendida y la aleatoria. Podemos ver que el resultado es prometedor. La heurística aprendida llega a ganar 10 partidas con 200 casos en la base de datos.

En la Figura 6.1 observamos de forma gráfica la evolución de la mejora de la heurística aprendida. Con 0 casos en la base de datos, el número de partidas ganadas es parecido al de partidas perdidas, ya que la heurística aprendida es muy parecida a la aleatoria. Conforme el número de partidas de la base de datos aumenta, la heurística aprendida se va imponiendo a la aleatoria. Vemos que a partir de 60 partidas la diferencia entre partidas ganadas y perdidas se hace muy significativa, contrarrestado levemente por las partidas que acaban en tablas.

Aunque las partidas ganadas no muestran un desarrollo estrictamente creciente (por ejemplo, para $x = 180$ hay una bajada inesperada, y también en las 20 primeras), si se observa en líneas generales la mejora de la heurística aprendida. La razón por la que existen lagunas en el crecimiento de las partidas ganadas, es debido a que el componente aleatorio de las partidas puede hacer que se den circunstancias especiales que hagan que localmente no se obtengan los resultados esperados teóricamente. Es de suponer que ocurra esto en experimentos de este tipo.

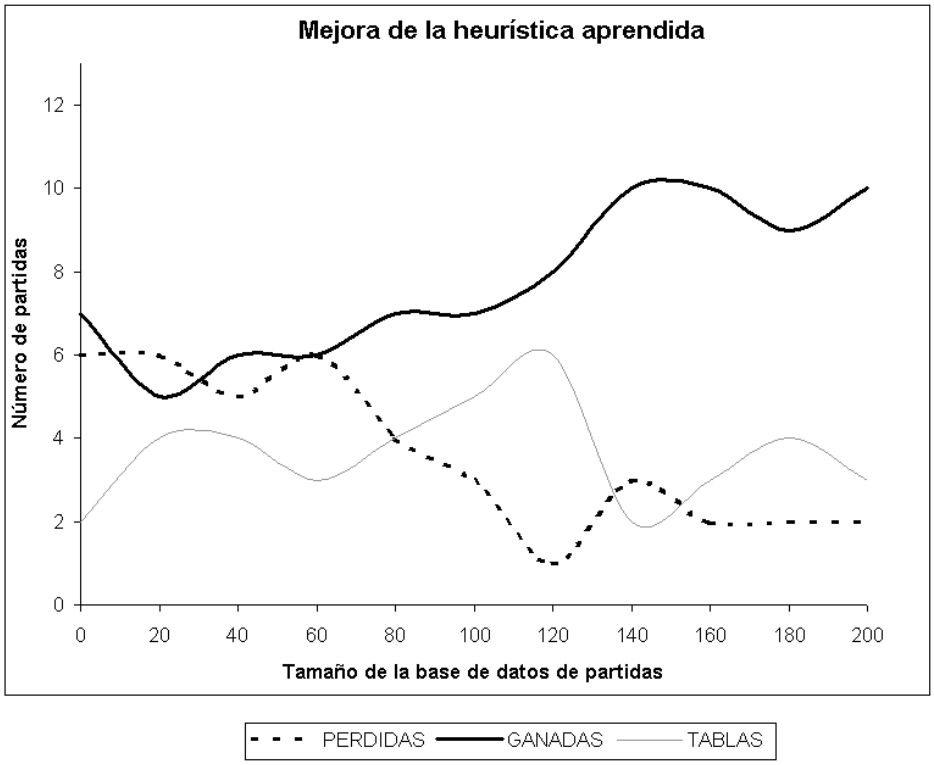

Figura 6.1: Mejora de la heurística aprendida

6.2. Evaluación de una posición de tablero

En este segundo experimento vamos a tomar una situación de partida y la evaluaremos de forma paulatina con las heurísticas fija, aleatoria y aprendida, conforme vamos aumentando el tamaño de la base de datos de partidas, de la misma forma que en el experimento anterior.

La Figura 6.2 muestra la situación de partida tomada como ejemplo. Claramente cualquier jugador de ajedrez divisa que las blancas tienen ventaja. En concreto disponen de un caballo y dos peones más, sin tener en cuenta otros aspectos, como la situación de las piezas. Esto supone unos -500 puntos según el valor de material de la heurística inicial (es negativo debido a que la utilidad se calcula tomando como referencia las negras). Pues bien, vamos a observar qué valor de utilidad asigna la heurística aprendida conforme aumentamos el tamaño de la base de datos. Indicar que el valor que da la heurística fija y aleatoria son independientes del tamaño de la base de datos, pero se mostrarán para una

mayor claridad.

En la Tabla 6.2 se muestra el valor de utilidad del tablero para los distintos tamaños de la base de datos de partidas y en la Figura 6.3 vemos de forma gráfica cómo el valor de utilidad que da la heurística aprendida se va acercando a un valor razonable (entorno a -500 puntos), mientras que la heurística aleatoria alterna valores sin ningún criterio, era de esperar.

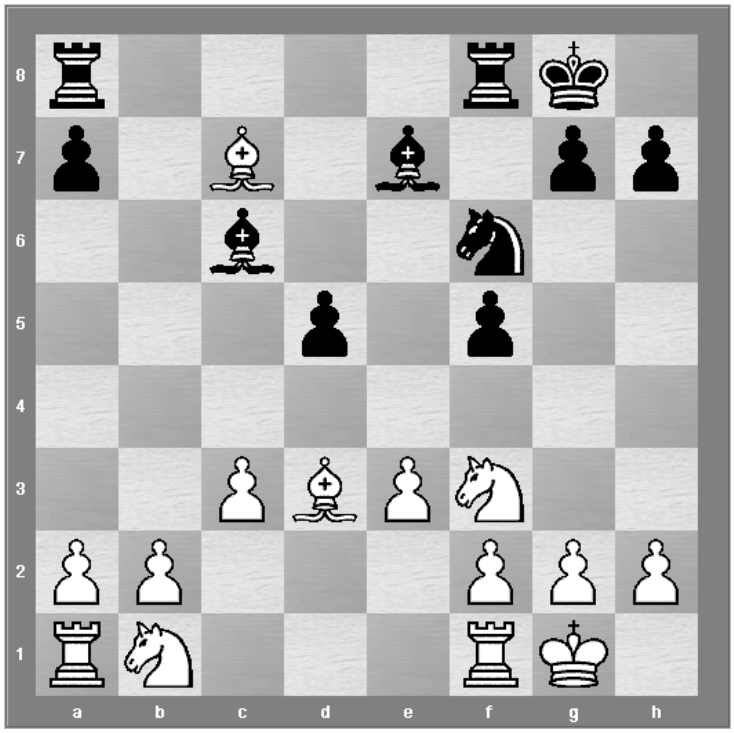

Figura 6.2: Tablero ejemplo para el experimento

Heurística	**0**	**20**	**40**	**60**	**80**	**100**	**120**	**140**	**160**	**180**	**200**
Inicial	-469	-469	-469	-469	-469	-469	-469	-469	-469	-469	-469
Aleatoria	-566	-426	-409	-421	-208	-563	-263	-730	-639	-694	-585
Aprendida	-243	-371	-373	-342	-353	-497	-439	-524	-506	-534	-531

Tabla 6.2: Valor de utilidad del tablero para los distintos tamaños de la base de datos de partidas

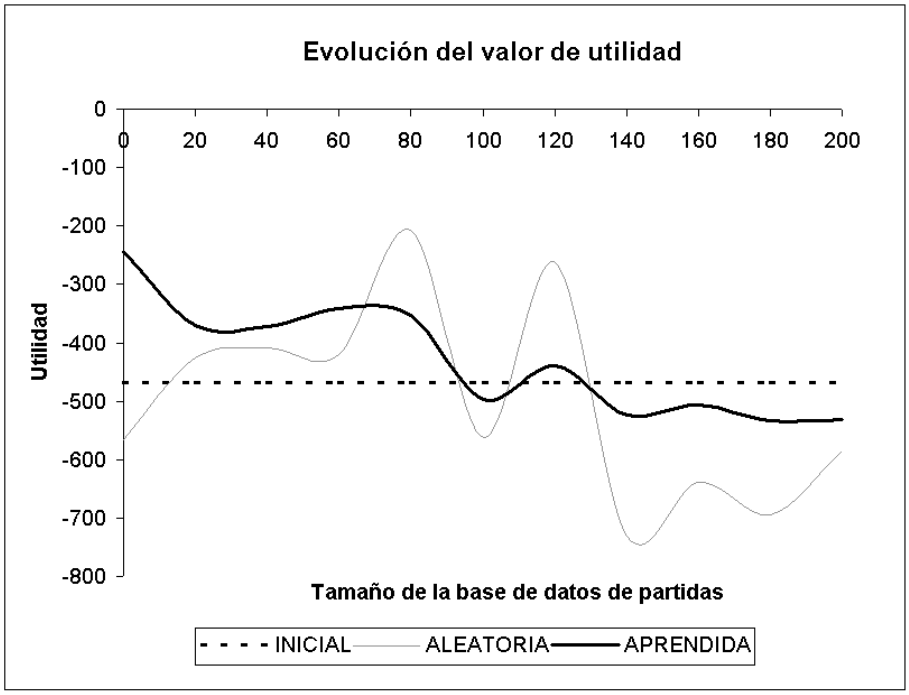

Figura 6.3: Evolución del valor de utilidad conforme aumentamos el tamaño de la base de datos

6.3. Conclusión

En este proyecto hemos desarrolado un programa de ajedrez que adapta su comportamiento al enemigo al que enfrenta y a su propia experiencia de juego. Los resultados de la experimentación indican que el aprendizaje tendrá mayor éxito cuanto más casos disponga la base de datos. El único inconveniente será que cuantas más partidas tenga ésta, más robusto será el clasificador de aprendizaje, es decir, para modificar las tablas de probabilidad de una forma sustancial, el número de casos deberá ser mucho mayor que en los comienzos del aprendizaje. En otras palabras, para conseguir un aprendizaje muy refinado de los parámetros de la heurística el número de partidas de la base de datos debe ser considerablemente grande.

Una solución a este problema podría ser comenzar con una base de datos vacía en un momento dado, con objeto de refinar la heurística aprendida hasta el momento.

En definitiva, pensamos que las redes bayesianas son una herramienta válida para construir sistemas adaptables al usuario, siguiendo una metodología parecida a la empleada en este proyecto.

Parte II

Apéndice

Capítulo 7

Notación algebraica en ajedrez

La FIDE (Federación Internacional de Ajedrez) sólo reconoce para sus propios torneos y matches un sistema de anotación, el Sistema Algebraico, y recomienda también el uso de este sistema uniforme de anotación ajedrecística para literatura y revistas de ajedrez. Las planillas en las que se utilice un sistema de anotación distinto al algebraico, no podrán ser utilizadas como prueba en casos en los que la planilla de un jugador se usa para tal fin. Un árbitro que observe que un jugador utiliza un sistema de anotación distinto al algebraico, advertirá al jugador acerca de este requisito.

7.1. Descripción del sistema algebraico

El sistema algebraico se define mediante una serie de reglas que se especifican a continuación:

1. Cada pieza es indicada por la primera letra, en mayúscula, de su nombre. Por ejemplo: R = rey, D = dama, T = torre, A = alfil, C = caballo.

2. Para la letra inicial del nombre de una pieza, cada jugador es libre de usar la primera letra del nombre utilizado normalmente en su país. Por ejemplo: A = alfil (en español); B = bishop (en ingles). En publicaciones impresas se recomienda el uso de figuras.

3. Cada movimiento de una pieza se indica por (a) la letra inicial del nombre de la pieza en cuestión y (b) la casilla de llegada. No hay guión entre (a) y (b). Por ejemplo:

a8	b8	c8	d8	e8	f8	g8	h8
a7	b7	c7	d7	e7	f7	g7	h7
a6	b6	c6	d6	e6	f6	g6	h6
a5	b5	c5	d5	e5	f5	g5	h5
a4	b4	c4	d4	e4	f4	g4	h4
a3	b3	c3	d3	e3	f3	g3	h3
a2	b2	c2	d2	e2	f2	g2	h2
a1	b1	c1	d1	e1	f1	g1	h1

Tabla 7.1: Identificación mediante letra y número de las 64 casillas del tablero

Ae5, Cf3, Td1. En el caso de peones, sólo se indica la casilla de llegada. Por ejemplo: e5, d4, a5.

4. Los peones no son indicados por su primera letra, sino que se les reconoce precisamente por la ausencia de la misma. Por ejemplo: e5, d4, a5.

5. Las ocho columnas (de izquierda a derecha para las blancas y de derecha a izquierda para las negras) son indicadas por las letras minúsculas: a, b, c, d, e, f, g y h, respectivamente.

6. Las ocho filas (de abajo arriba para las blancas y de arriba abajo para las negras) están numeradas: 1, 2, 3, 4, 5, 6, 7 y 8, respectivamente. Consecuentemente, en ha posición inicial las piezas y peones blancos se colocan en la primera y segunda filas; las piezas y peones negros en la octava y séptima filas.

7. Como consecuencia de las reglas anteriores, cada una de las 64 casillas está indicada invariablemente por una sola combinación de una letra y un número.

8. Cada movimiento de una pieza se indica por (a) la letra inicial del nombre de la pieza en cuestión y (b) la casilla de llegada. No hay guión entre (a) y (b). Por ejemplo: Ae5, Cf3, Td1. En el caso de peones, sólo se indica la casilla de llegada. Por ejemplo: e5, d4, a5.

9. Cuando una pieza realiza una captura, se inserta una x entre (a) la letra inicial del nombre de la pieza en cuestión y (b) la casilla de llegada. Por ejemplo: Axe5, Cxf3, Txd1.

10. Cuando un peón realiza una captura, debe indicarse la columna de partida, luego una x y finalmente la casilla de llegada. Por ejemplo: dxe5, gxf3, axb5. En el caso de una captura al paso, se pone como casilla de llegada la que finalmente ocupa el peón que realiza la captura, añadiéndose a la anotación a.p. Ejemplo: exd6 a.p..

11. Si dos piezas idénticas pueden moverse a la misma casilla, la pieza que se mueve se indica como sigue:

    - Si ambas piezas están en la misma fila: por (a) la letra inicial del nombre de la pieza, (b) la columna de la casilla de salida y (c) la casilla de llegada.
    - Si ambas piezas están en la misma columna: por (a) la letra inicial del nombre de la pieza, (b) la fila de la casilla de salida y (c) la casilla de llegada.
    - Si las piezas están en filas y columnas distintas, es preferible el método (1). En el caso de una captura, debe insertarse una x entre (b) y (c).

    Ejemplos:

    - Hay dos caballos, en las casillas g1 y e1, y uno de ellos mueve a la casilla f3: según el caso, puede ser Cgf3 o Cef
    - Hay dos caballos, en las casillas g5 y g1, y uno de ellos mueve a la casilla f3: según el caso, puede ser C5f3 o C1f3.
    - Hay dos caballos, en las casillas h2 y d4, y uno de ellos mueve a la casilla f3: según el caso, puede ser Chf3 o Cdf3.

    Si se da una captura en la casilla f3, se modifican los ejemplos anteriores insertando una x: según el caso, puede ser Cgxf3 o Cexf3, C5xf3 o C1xf3, Chxf3 o Cdxf3, respectivamente.

12. Si dos peones pueden capturar la misma pieza o peón del adversario, el peón que se mueve se indica por (a) la letra de la columna de salida, (b) una x, (c) la casilla de

llegada. Por ejemplo: si hay peones blancos en las casillas c4 y e4 y un peón o pieza negro en la casilla d5, la anotación para la jugada de Blancas puede ser, según el caso: cxd5 o exd5.

13. En el caso de la promoción de un peón, se indica el movimiento efectivo del peón, seguido inmediatamente por el símbolo = y la letra inicial de la nueva pieza o directamente poniendo la letra inicial de la pieza sin símbolo =. Por ejemplo: d8D, f8=C, b1=A, g1T.

14. La oferta de tablas se anotará como (=).

Las abreviaturas utilizadas son:

- 0-0: enroque con la torre de h1 o de h8 (enroque por el flanco de rey o enroque corto).
- 0-0-0: enroque con la torre de a1 o de a8 (enroque por el flanco de dama o enroque largo).
- x: captura
- +: jaque
- ++ ó # : jaque mate
- a.p.: captura de peón al paso.

7.2. Ejemplo de partidas

Notación española:

1. d4 Cf6 2. c4 e6 3. Cc3 Ab4 4. Ad2 0-0 5. e4 d5 6. exd5 exd5 7. cxd5 Axc3 8. Axc3 Cxd5 9. Cf3 b6 10. Db3 Cxc3 11. bxc3 c5 12. Ae2 cxd4 13. Cxd4 Te8 14. 0-0 Cd7 15. a4 Cc5 16. Db4 Ab7 17. a5

Notación inglesa:

1. d4 Nf6 2. Bg5 d5 3. e3 g6 4. Bxf6 exf6 5. c4 dxc4 6. Bxc4 Bd6 7. Nc3 O-O 8. h4 h5 9. Qc2 Kg7 10. Nge2 Nd7 11. O-O-O Nb6 12. Bb3 Qe7 13. Qe4 Re8 14. Qxe7 Bxe7 15. Nf4 a5 16. a3 Bd6 17. Nd3 a4 18. Ba2 Ra5 19. e4 Rd8 20. Rhe1 Be7 21. Nf4 f5 22. e5 Bxh4 23. g3 Bg5 24. d5 c6 25. dxc6 Rxd1+ 26. Kxd1 bxc6 27. e6 Bxf4 28. e7 Bd7 29. e8=Q Bxe8 30. Rxe8 Bd6 31. Ne2 Rb5 32. Kc2 f4 33. gxf4 Rf5 34. Re4 h4 35. Ng1 Rxf4 36. Rxf4 Bxf4 37. b3 f5 38. bxa4 Nxa4 39. Nh3 Bd6 40. f4 Nb6 41. Kb2 Kh6 42. Bf7 Nd5 43. Be8 Nxf4 44. Nxf4 Bxf4 45. Bxc6 g5

7.3. Registro de las partidas jugadas

Por cada partida que se juega se genera la notación algebraica correspondiente y se añade al archivo *log.pgn* situado en el mismo directorio de la aplicación. La notación usada no es estrictamente la anterior, pero se considera válida tambíén. Lo único que cambia es que los movimientos se especifican por la casilla de inicio y de fin separadas por un guión. Observamos en la Figura 7.1 un ejemplo de partida. Vemos como existe previo a los movimientos una cabecera con información acerca de los jugadores, lugar de juego, resultado, fecha, etc.

[Event ”PFC”]
[Site ”UAL”]
[Date ”2005.02.24”]
[Round ”1”]
[White ”BayessChess aleatorio”]
[Black ”BayesChess fijo”]
[Result ”1-0”]
[ECO ”A00”]
[PlyCount ”11”]
[EventDate ”2005.02.24”]

1. b1-c3 c7-c6 2. d2-d4 d8-b6 3. a2-a3 h7-h5 4. e2-e4 d7-d6 5. f1-e2 h5-h4 6. b2-b4 b8-d7 7. c3-a4 b6-d8 8. c1-g5 f7-f6 9. e2-h5+ h8xh5 10. d1xh5+ g7-g6 11. h5xg6 #

Figura 7.1: Ejemplo de partida almacenada en el registro de jugadas

Capítulo 8

Aspectos de diseño e implementación

8.1. Diagrama de clases

En la Figura 8.1 podemos observar el diagrama de clases completo de la aplicación, y para más claridad aparece dividido en cuatro partes:

- Figura 8.2: Cuadrante superior izquierdo
- Figura 8.3: Cuadrante superior derecho
- Figura 8.4: Cuadrante inferior izquierdo
- Figura 8.5: Cuadrante inferior derecho

8.2. Estructuras de datos

Ya se ha mostrado en el diagrama de clases la estructura de la aplicación, pero resulta conveniente comentar algunas estructuras de datos que considero de importancia.

- **Tablero y piezas**: La estructura de datos utilizada para el tablero de ajedrez y sus piezas ha sido un array unidimensional de 64 posiciones (número de casillas de un tablero de ajedrez) en donde cada una de ellas almacena un número que identifica a una pieza y -1 una casilla libre. En la Figura 8.6 y en la Figura 8.7 podemos observar el array que simula el tablero y la numeración de las piezas. Indicar que en un principio se usó un array bidimensional para la representación del tablero y,

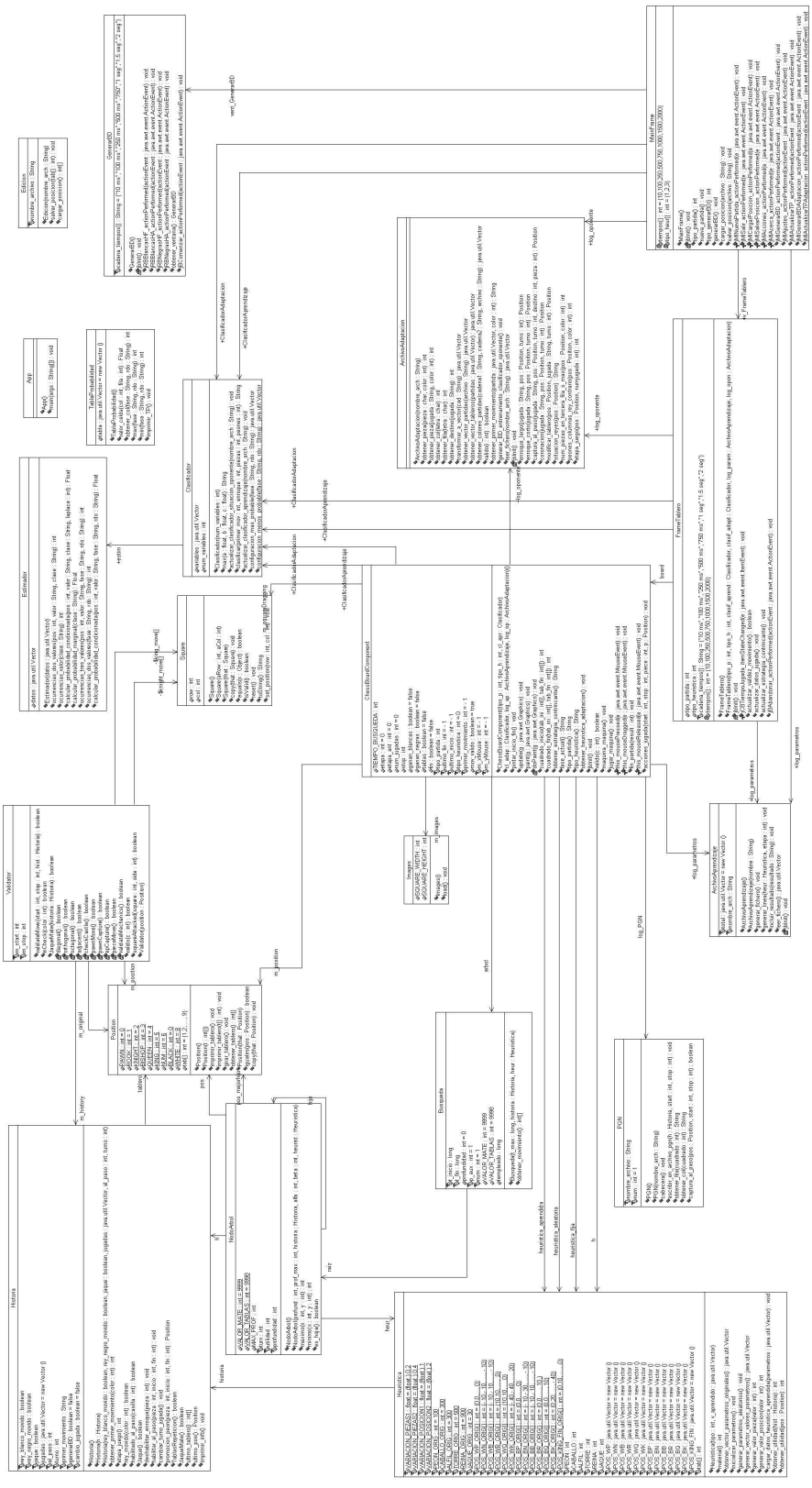

Figura 8.1: Diagrama de clases completo de la aplicación

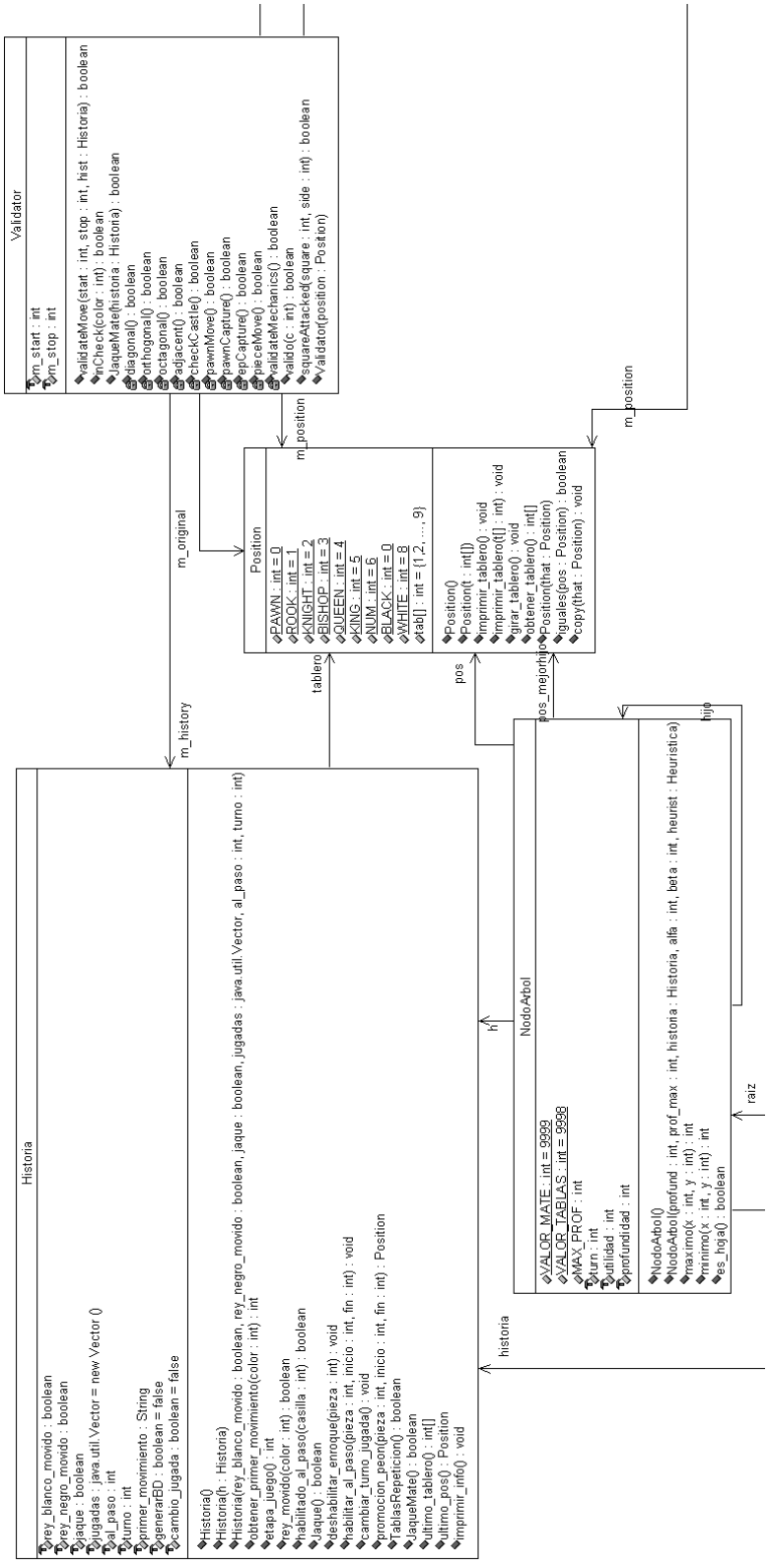

Figura 8.2: Diagrama de clases de la aplicación. Parte I

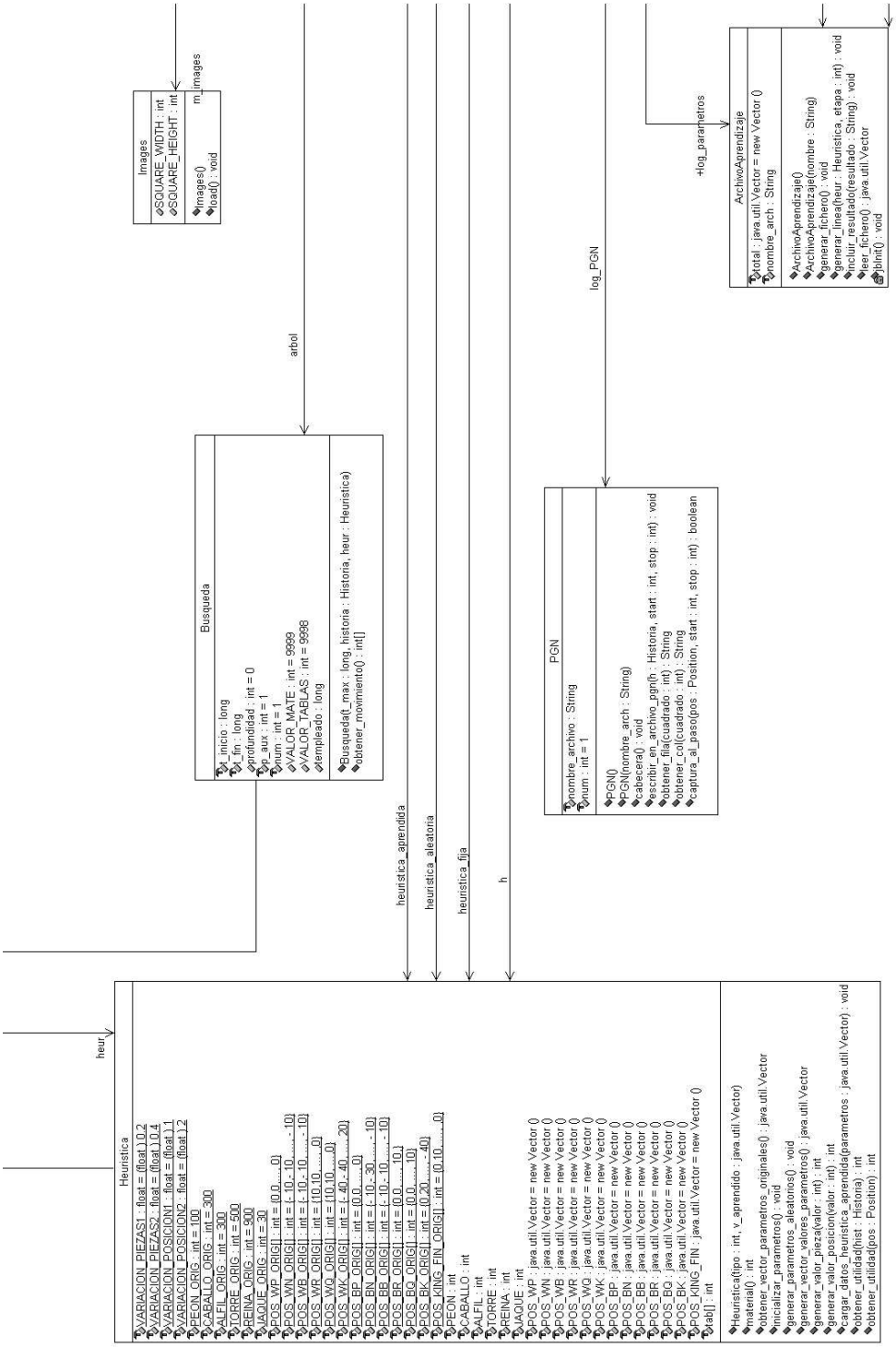

Figura 8.3: Diagrama de clases de la aplicación. Parte II

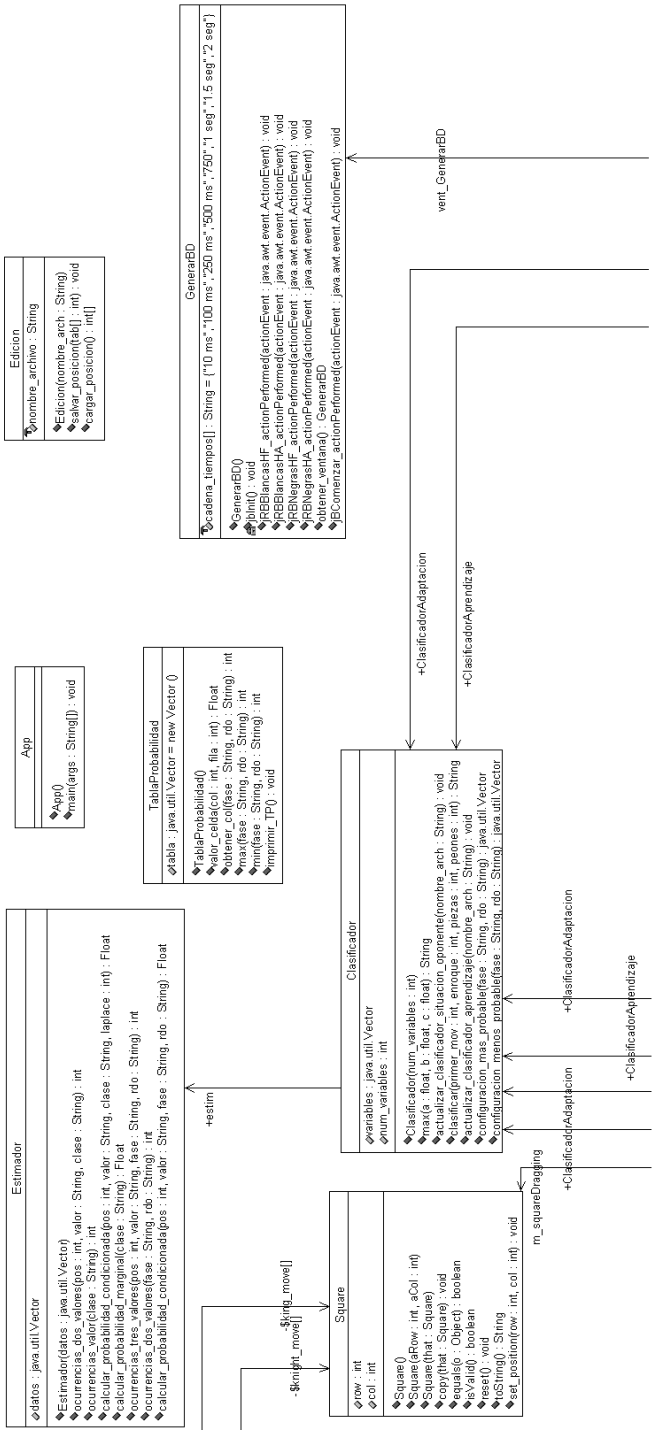

Figura 8.4: Diagrama de clases de la aplicación. Parte III

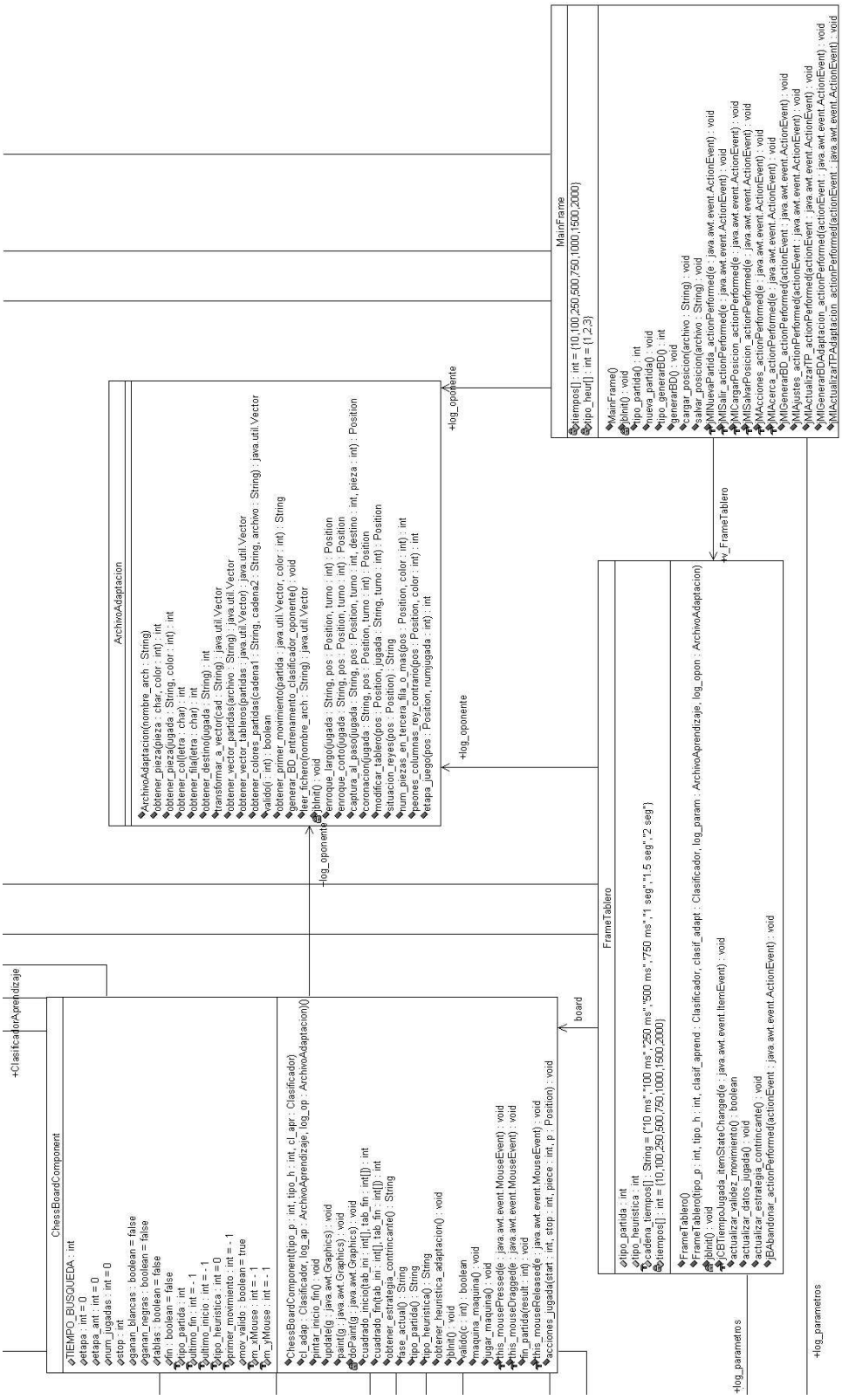

Figura 8.5: Diagrama de clases de la aplicación. Parte IV

posteriormente, opté por usar uno unidimensional por razones de eficiencia en el algoritmo de búsqueda.

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

Figura 8.6: Estructura de datos del tablero

- **NodoArbol**: Es la estructura usada en el árbol de búsqueda de la jugada óptima. Se trata de una clase con información del nodo actual (situación del tablero, jaque, enroque habilitado, peon al paso habilitado, ...) y un vector en donde se almacenan sus hijos, que vuelven a ser estructuras NodoArbol. El nodo raíz será él inicio de la estructura de árbol.
- **Cuadrado**: Estructura para representar una casilla del tablero. Sus atributos más importantes son la fila y columna de la casilla en cuestión.

NEGRAS

1	2	3	4	5	3	2	1
0	0	0	0	0	0	0	0
-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1
8	8	8	8	8	8	8	8
9	10	11	12	13	11	10	9

BLANCAS

Figura 8.7: Numeración de las piezas sobre el tablero

Capítulo 9

Manual de usuario

9.1. Ejecución del programa

Para la ejecución del programa es necesario tener instalado el *JDK (Java Development Kit)* en su versión *1.4.2.* que es la que he utilizado, o como mínimo su versión reducida *JRE (Java Runtime Environment)* destinada a ejecutar código java (no permite compilar).

Una vez realizado esto, se debe editar el archivo *makefile.bat* localizado en el directorio del proyecto e incluir la ruta o *path* en donde se encuentran instalados los archivos *javac.exe* y *java.exe* , para que el proceso de compilación y ejecución tenga éxito, o en su defecto sólo ejecución. En la Figura 9.1 se puede observar el contenido del archivo *makefile.bat*. La primera línea especifica la ruta de los archivo *java.exe* y *javac.exe*. La segunda compila los archivos fuente (si no se desea realizar esta acción se debe disponer de los archivos *.class* para la ejecución). Posteriormente se ejecuta la aplicación especificando límite inferior y superior de la utilización de la memoria del ordenador. Por último se borran todos los archivos *.class* a fin de que, ante una modificación en los archivos fuente, sean de nuevo generados sin sobreescribir a los anteriores y evitarnos problemas.

9.2. Descripción de la interfaz

La interfaz de la aplicación mostrada en la Figura 9.2 se compone de tres partes bien diferenciadas. En primer lugar, una parte destinada al juego en sí, en donde se puede cargar una partida en un formato determinado (véase punto 3.3 del apéndice), así como salvarla y por supuesto jugar. Otra parte que engloba el proceso de aprendizaje, en donde

```
path C:\j2sdk1.4.2_08\bin

javac *.java

java -Xms100M  -Xmx500M App

del *.class
```

Figura 9.1: Contenido del archivo *makefile.bat*

se puede generar una base de datos de partidas, así como actualizar las tablas de probabilidad del clasificador de aprendizaje. La tercera parte es la encargada del clasificador de estrategia, y realiza las mismas acciones que el clasificador anterior pero aplicado a éste.

Indicar que al arrancar la aplicación se actualizan ambos clasificadores, por lo que no será necesario hacerlo durante la partida, aunque se puede hacer sin ningún problema.

9.2.1. Menú *Acciones*

En el menú *Acciones* mostrado en la Figura 9.3 tenemos varias alternativas: jugar una nueva partida, cargar una posición de tablero, salvar una posición de tablero o salir del programa.

9.2.1.1. *Nueva partida*

Si seleccionamos esta opción la aplicación mostrará la ventana de la Figura 9.4 y tendremos que especificar el tipo y características de la partida que queremos jugar.

En primer lugar, se escoge quién juega con blancas, si el usuario o la máquina. Al seleccionar una opción en un color se actualiza el otro con la opción contraria. Se puede indicar el nombre de los jugadores en un espacio de texto habilitado para ello.

Por otro lado, debemos especificar el tiempo de búsqueda que queremos que como mínimo la máquina piense una jugada. Evidentemente cuánto más tiempo seleccionemos mayor calidad tendrá la jugada, aunque también tardará mucho más tiempo. El valor

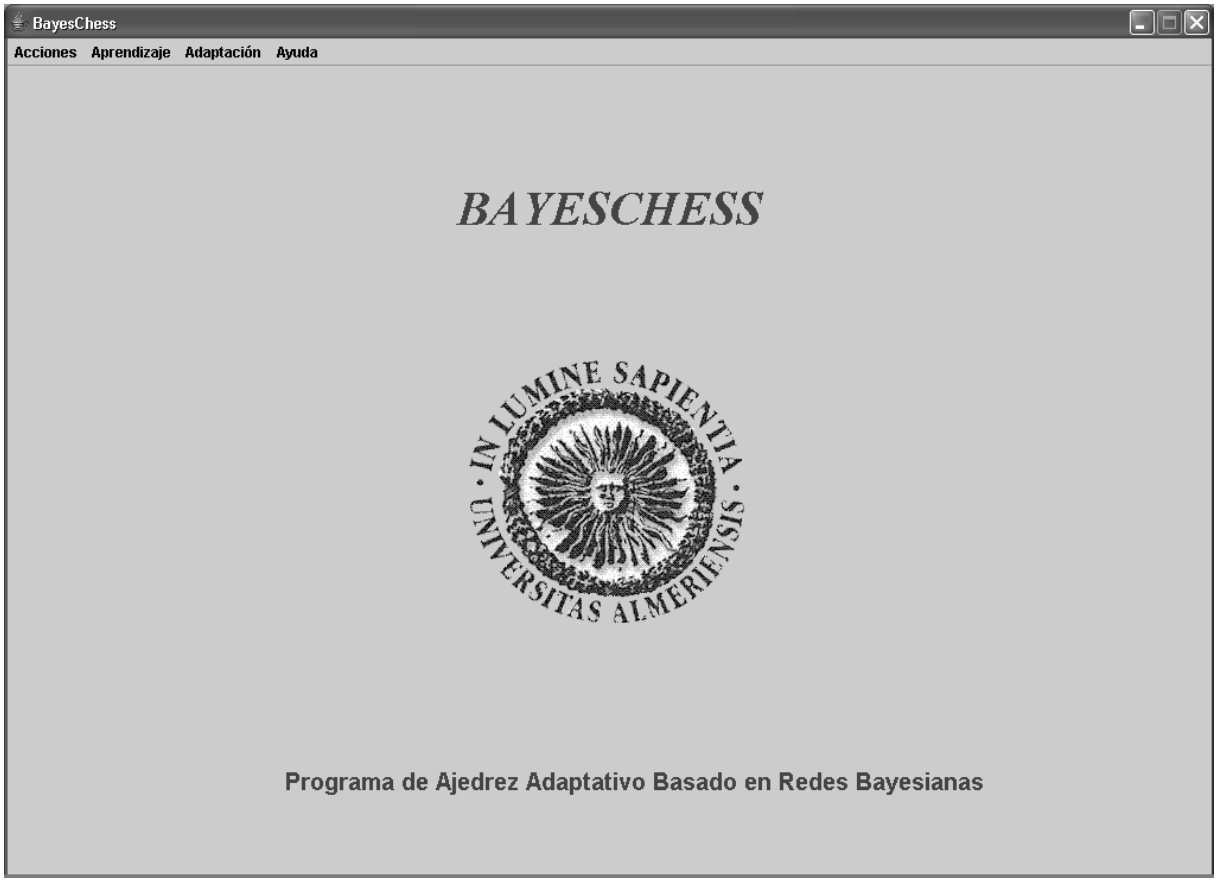

Figura 9.2: Interfaz principal de la aplicación

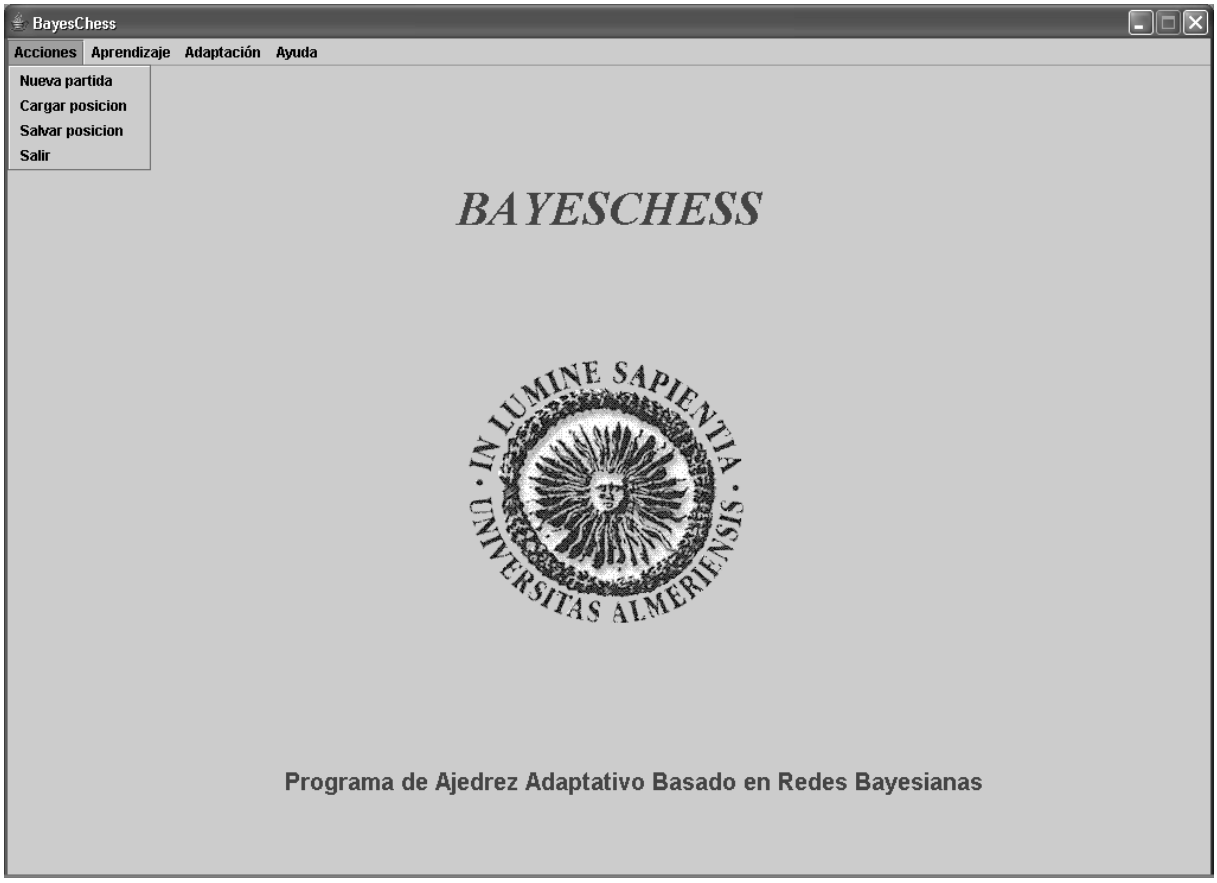

Figura 9.3: Menú *Acciones*

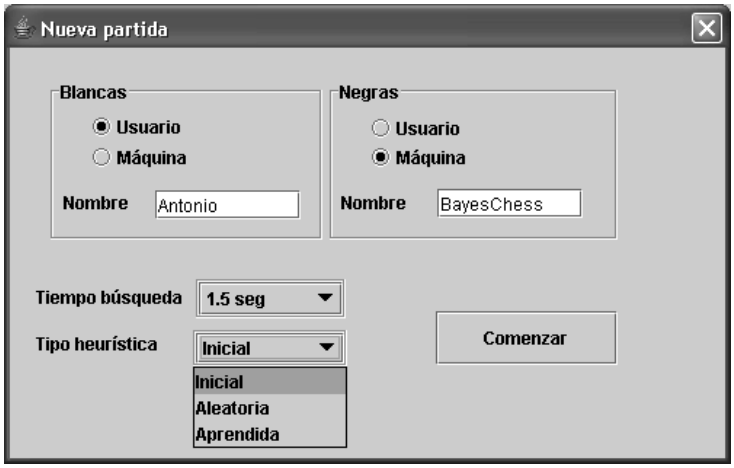

Figura 9.4: Menú *Comienzo de una nueva partida*

puede ir desde 10 milisegundos hasta 2 segundos.

Por último seleccionaremos el tipo de heurística que utilizará la máquina en su juego. Hay tres posibles:

- **Inicial**: Es la heurística con la que inicialmente he implementado el ajedrez. Si seleccionamos esta alternativa el motor de ajedrez usará siempre la misma configuración de parámetros ante situaciones idénticas.

- **Aleatoria**: Este tipo de heurística genera unos parámetros aleatorios (tomando como referencia los iniciales) para seleccionar el movimiento adecuado en cada situación. En ocasiones, será buena la heurística generada, y en otras no tanto.

- **Aprendida**: Se trata de la heurística procedente del clasificador de aprendizaje, es decir, está basada en las partidas de la base de datos de entrenamiento. Cuantas más jugadas se incluyan en dicha base de datos más calidad tendrá el juego de este tipo de heurística.

Una vez realizados estos pasos, pulsamos el botón *Comenzar* y se mostrará una ventana con el tablero de juego preparado para que el usuario comience. Para realizar un movimiento hay que pulsar con el botón izquierdo del ratón en la pieza deseada, arrastrarla hasta el destino y soltarla allí. En la Figura 9.5 se muestra el aspecto de esta ventana en una partida aún no finalizada.

Podemos observar que además del tablero con las piezas, hay a la derecha cuatro zonas diferenciadas. Comenzando por la parte superior se muestran los datos de la partida, esto es, el tipo de heurística seleccionada en la ventana anterior y el tiempo mínimo de búsqueda del mejor movimiento, característica que puede ser modificada durante la partida.

Posteriormente, tenemos los datos de la última jugada: fase de la partida, profundidad alcanzada en al árbol de búsqueda, tiempo empleado en la búsqueda, utilidad actual-

forma de indicar numéricamente quién tiene ventaja actualmente según la heurística de juego de la máquina en esta partida; si es positiva las negras tienen ventaja, si es negativa, las blancas- y por último una indicación explícita de quién lleva ventaja en el juego (BayessChess o Usuario).

A continuación se informa de la estrategia que está siguiendo el usuario en el instante actual. Puede tomar tres valores: atacante, defensiva o mixta.

Por último se muestra cuál es la respuesta de la máquina a la estrategia que sigue el usuario. Hay dos alternativas: maximizar la probabilidad de ganar o minimizar la probabilidad de perder.

Se ha incluido, además, un botón para abandonar la partida. Si lo pulsamos se cerrará la ventana actual y se adjudicará la partida a la máquina.

En la Figura 9.6 podemos observar una partida finalizada y el resultado de la misma situado en el lugar en donde durante la partida se especifica de quién es la ventaja. Las posibles alternativas son: ganan las negras, ganan las blancas o tablas.

9.2.1.2. *Cargar posición*

Esta opción del programa carga una posición de tablero previamente guardada o creada por nosotros *a mano*. Esta opción es muy útil cuando queremos reproducir una partida a partir de una posición en concreto.

9.2.1.3. *Salvar posición*

Pulsando esta opción en cualquier momento de la partida se almacena en un archivo el tablero actual quedando disponible para un futuro uso.

9.2.1.4. *Salir*

Finaliza la ejecución de la aplicación.

Figura 9.5: Partida en juego

Figura 9.6: Partida finalizada

9.2.2. Menú *Aprendizaje*

A traves de este menú (Figura 9.7) se puede generar una base de datos de jugadas y actualizar las tablas de probabilidad del clasificador para el aprendizaje.

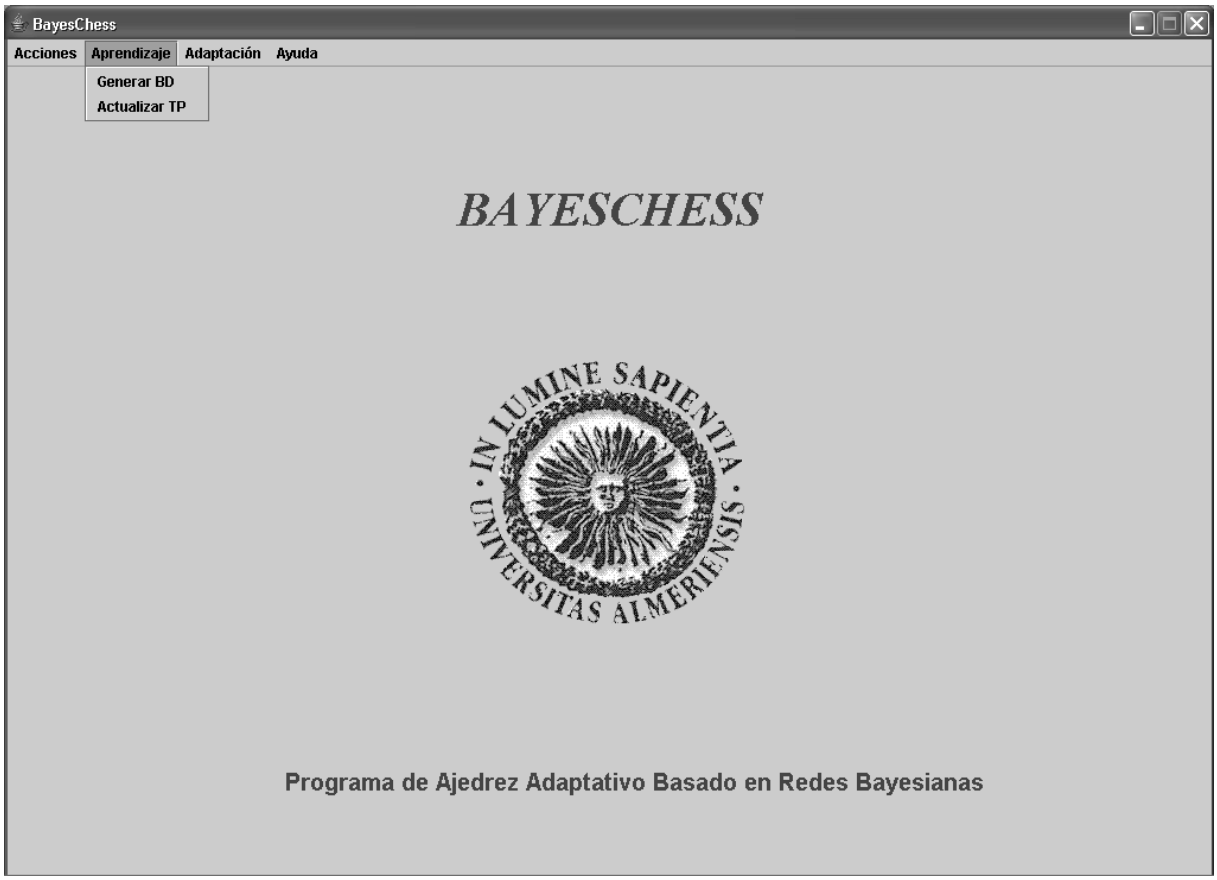

Figura 9.7: Menú *Aprendizaje*

9.2.2.1. *Generar base de datos*

Pulsando sobre esta opción se activará la ventana de la Figura 9.8 dándonos la opción de que la máquina juegue un número determinado de partidas con ella misma, usando cada color un tipo de heurística determinada. Dicho heurística debe ser especificada en esta ventana, así como el número de partidas y el tiempo de búsqueda. Se generará un archivo de nombre *parámetros.txt* en el directorio de la aplicación.

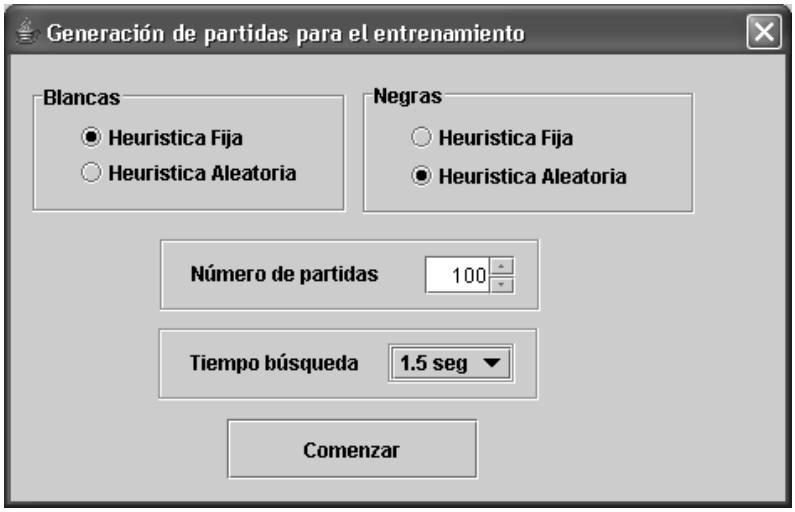

Figura 9.8: Ventana para la generación de la base de datos de partidas

9.2.2.2. *Actualizar tablas de probabilidad*

Si seleccionamos esta opción se actualizarán las tablas de probabilidad del clasificador para el aprendizaje a partir de la base de datos de partidas generada anteriormente.

9.2.3. Menú *Adaptación*

A traves de este menú (Figura 9.9) se puede generar una base de datos de jugadas y actualizar las tablas de probabilidad del clasificador para la adaptación.

9.2.3.1. *Generar base de datos*

Pulsando este botón se generará una base de datos con valores de las variables necesarias para generar el clasificador de adaptación. Serán necesarios los archivos *fischerkasparov.pgn*, *karpov.pgn*, *illescas.pgn* y el archivo generado tras la generación de la base de datos será *oponente.txt*, todos ellos localizados en el directorio de la aplicación.

9.2.3.2. *Actualizar tablas de probabilidad*

A partir de la base de datos incluida en el archivo *oponente.txt* se actualizará las tablas de probabilidad del clasificador

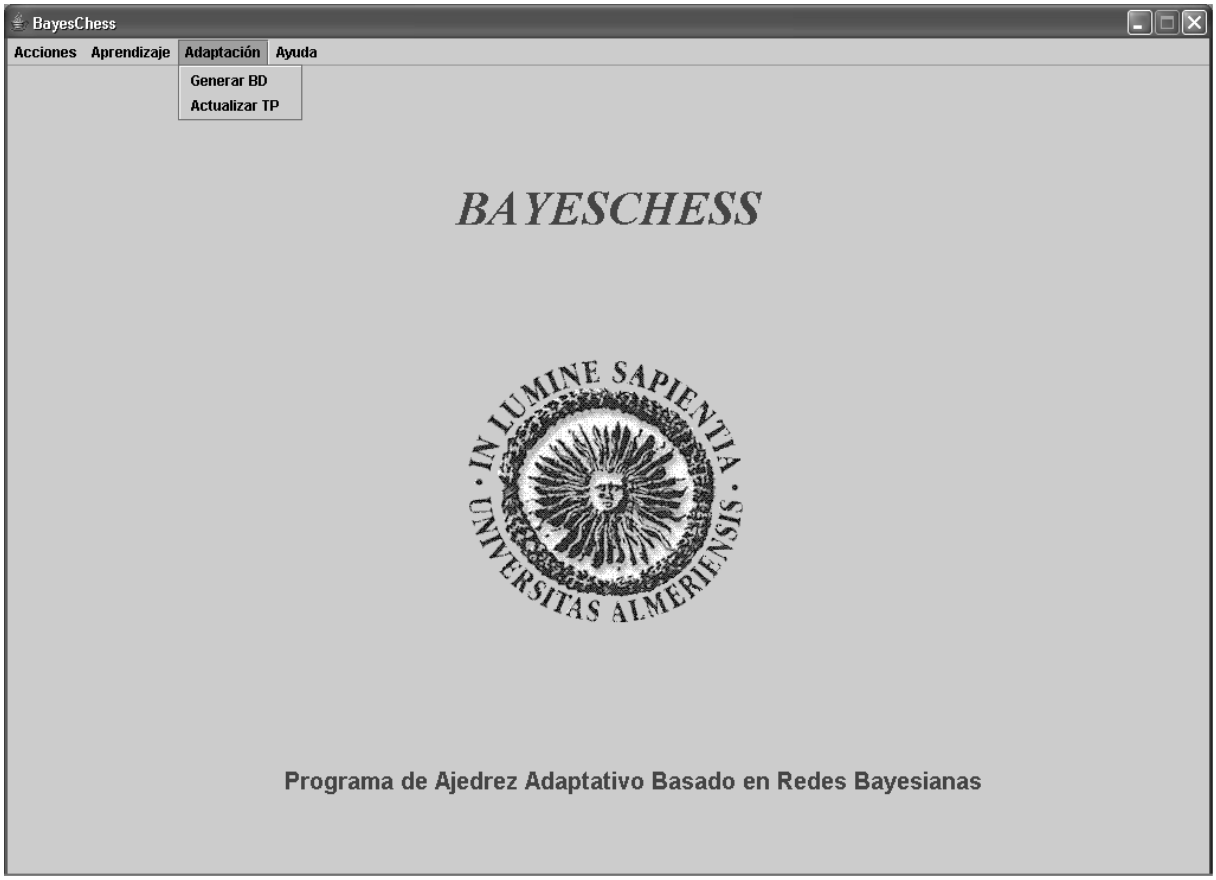

Figura 9.9: Menú *Adaptación*

9.2.4. Menú *Ayuda*

Se muestra este menú en la Figura 9.10.

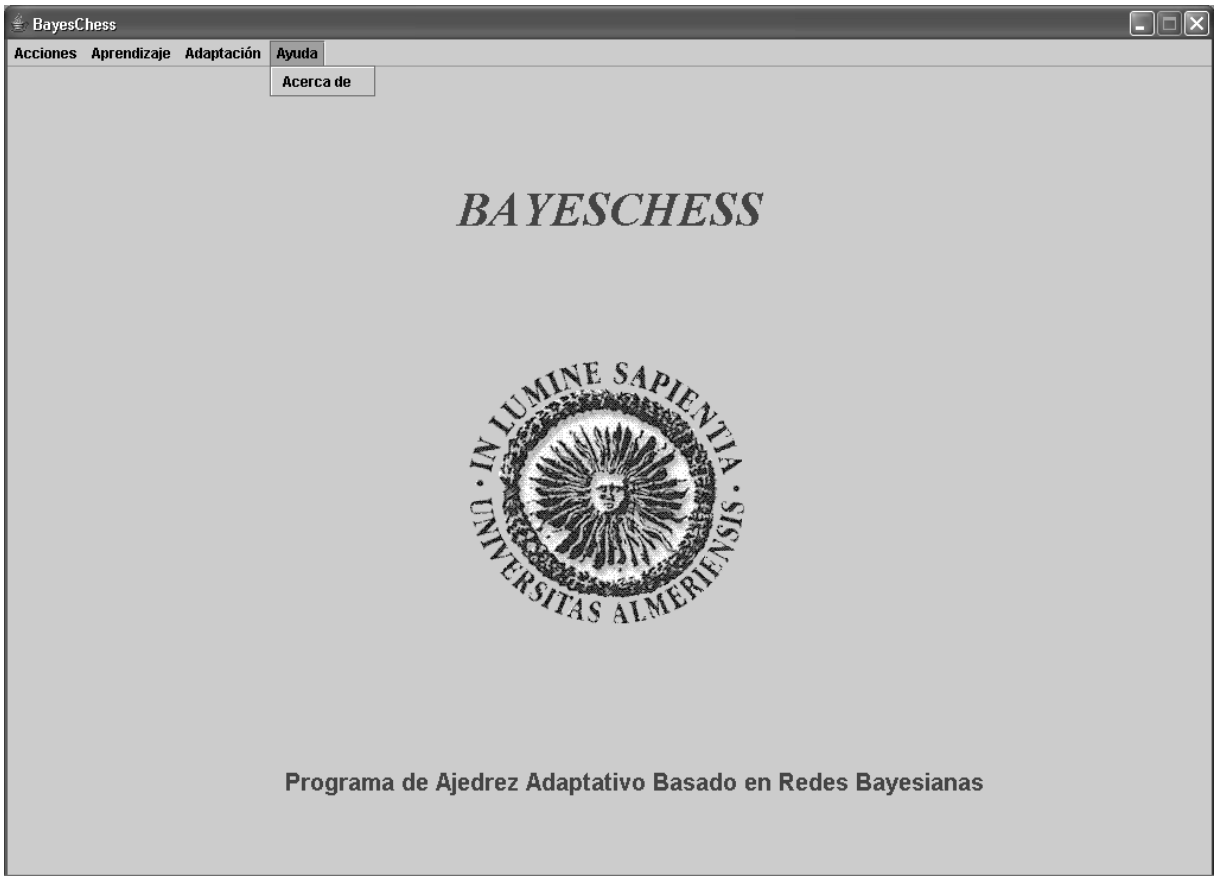

Figura 9.10: Menú *Ayuda*

9.2.4.1. *Acerca de ...*

En la Figura 9.11 se informa del nombre y versión del programa

Figura 9.11: Ventana *Acerca de...*

9.3. Formato del archivo para cargar y salvar posiciones de tablero

Se ha utilizado para ello un archivo con extensión *txt* en donde se almacenan una serie de caracteres que definen una posición de tablero en concreto. En la Figura 9.12 se puede observar el tablero inicial según el formato que he seguido. Como podemos ver hay ocho filas y ocho columnas. Cada posición del tablero se identifica con dos caracteres, cuyo significado se especifica a continuación:

- BR: Abreviatura en inglés de *Black Rook*
- BN: Abreviatura en inglés de *Black Knigth*
- BB: Abreviatura en inglés de *Black Bishop*
- BQ: Abreviatura en inglés de *Black Queen*
- BK: Abreviatura en inglés de *Black King*
- BP: Abreviatura en inglés de *Black Pawn*
- WR: Abreviatura en inglés de *White Rook*
- WN: Abreviatura en inglés de *White Knigth*
- WB: Abreviatura en inglés de *White Bishop*
- WQ: Abreviatura en inglés de *White Queen*
- WK: Abreviatura en inglés de *White King*
- WP: Abreviatura en inglés de *White Pawn*
- - -: Casilla vacía

```
BR  BN  BB  BQ  BK  BB  BN  BR
BP  BP  BP  BP  BP  BP  BP  BP
--  --  --  --  --  --  --  --
--  --  --  --  --  --  --  --
--  --  --  --  --  --  --  --
--  --  --  --  --  --  --  --
WP  WP  WP  WP  WP  WP  WP  WP
WR  WN  WB  WQ  WK  WB  WN  WR
```

Figura 9.12: Formato del archivo de posición

Bibliografía

[1] Castillo, E., Gutiérrez, J.M., Hadi, A.S. (1996). *Sistemas expertos y modelos de redes probabilísticas*. Monografías de la Academia de Ingeniería.

[2] Hernández, J., Ramírez, M.J., Ferri, C. (2004). *Introducción a la minería de datos*. Editorial Pearson.

[3] Fürnkranz, J.(1996). Machine Learning In Computer Chess: The Next Generation. *International Computer Chess Association Journal*.

[4] Gámez, J.A., Puerta, J.M. (1998). *Sistemas expertos probabilísticos. Colección Ciencia y Técnica*. Ediciones de la Universidad de Castilla-La Mancha.

[5] Jensen, F.V. (1996). *An introduction to Bayesian networks*. UCL Press.

[6] Nilsson, N. (2004). *Inteligencia artificial: una nueva síntesis*. Editorial McGraw Hill.

[7] Russel, S., Norvig, P. (2004). *Inteligencia Artificial, un enfoque moderno, 2ª Edición*. Editorial Prentice Hall.

[8] Pachman, L., Kühnmund, V. *Ajedrez y computadores*. Colección Escaques.

[9] Cascales, Lucas, Mira, Pallarés, Sánchez-Pedreño. (2003). *El libro de LaTeX*. Prentice Hall.

[10] Armañanzas, Rubén. (2004). *Proyecto Fin de Carrera: Medidas de filtrado de selección de variables mediante la plataforma "Elvira"*.

[11] Li, L.(1998). *Java: Data Structures and Programming*. Editorial Springer.

[12] Deitel y Deitel. (1998). *Cómo programar en Java*. Editorial Prentice Hall.

[13] Borrajo, D., Juristo, N., Martínez, V., Pazos, J. (1997). *Inteligencia Artificial: Métodos y técnicas*. Centro de estudios Ramón Areces S.A.

[14] Ponce, L. (2004). *Ajedrez en 20 lecciones para principiantes*. Editorial de Vecchi.

[15] `http://www.mallardsoft.com/chessboard.htm`. Código fuente en Java de la interfaz y validación de movimiento en el juego del ajedrez

[16] `http://java.sun.com/`. Página de Java Sun

[17] `http://www.princeton.edu/ jedwards/cif/intro.html`. Información muy completa sobre el ajedrez: aperturas, tácticas, finales de juego, grandes partidas, reglas básicas de juego. Además tiene la posibilidad de jugar una partida de ajedrez.

[18] `http://www.redhotpawn.com/rival/programming/evaluation.php`. Página en donde se especifica de forma sencilla la importancia numérica del material en ajedrez.

[19] `http://www.cs.ucf.edu/ dmarino/ComputerChess.ppt`. Archivo que explica las técnicas de programación de ajedrez de forma razonada

[20] `http://www.stedwards.edu/science/baker/help/javamakefile.html`. Página web en donde se detalla cómo hacer un archivo *makefile* para la compilación y ejecución de programas Java

[21] `http://www.chiesu.com/`. Arte y Ajedrez

[22] `http://www.gamedev.net/reference/programming/features/chess4/page5.asp`. Todo lo que hay que saber sobre la búsqueda en ajedrez

[23] `http://www.chesslive.de/`. Base de datos de partidas online con buscador de posiciones, muy interesante

[24] `http://www.chess-poster.com/spanish/pgn/archivos_pgn.htm`. Bases de datos de partidas de los grandes maestros en formato PGN

[25] `http://user.cs.tu-berlin.de/ huluvu/WinFIG.htm`. Página del software WinFig muy útil para crear cualquier gráfico y exportarlo a formato *eps*, entre otros muchos.

[26] http://www.pgnmentor.com. Un excelente software que permite la búsqueda, visualización y creación de archivos de partidas en PGN files.

[27] http://www.fide.com/. Federación mundial de ajedrez

[28] http://www.fadajedrez.com/. Federación Andaluza de Ajedrez

[29] http://www.feda.org/. Federación española de ajedrez