Universidad Carlos III de Madrid

Escuela politécnica Superior

Ingeniería Técnica en informática de Gestión

PROYECTO DE FIN DE CARRERA

Implementación del juego Camelot aplicando algoritmos de búsqueda intensiva

Alumno: Ana Chimeno Laíño

Tutor: Carlos Linares López

Agradecimientos

Gracias a Carlos por su espíritu incansable, por hacerme salir después de cada reunión con fuerzas renovadas y ánimos de ganar al campeón mundial de Camelot; por todo lo que he aprendido de él, y todo lo que he disfrutado con este proyecto.

Gracias a mis padres, que aunque no sepan lo que es un algoritmo recursivo, me han preguntado, me han apoyado, me han agobiado, me han regañado y me han aguantado en todo mi proyecto y mi carrera, y sin los que nada de esto, me hubiera sido posible realizar. Y a mi hermano Javi, mi mejor amigo, que aunque no les guste programar, también me ha ayudado a su manera en este proyecto, siempre ahí, incondicional.

Gracias a Bea por todo, desde que decidimos matricularnos en esta universidad, hasta ahora mismo, que ya acabamos, cada paseo, cada charla y cada momento. A Víctor, por ser el Ying de mi Yang y a veces el Yang de mi Ying. A Alexandra, por su apoyo, que con su experiencia personal me han motivado a seguir dándolo todo en cada instante. A Silvi, porque en realidad no sabe cuánto me ayuda "desconectar" con ella para cargar las pilas. Y a toda la gente maravillosa que he conocido en la universidad, informáticos y "telecos", por todos esos ratos dentro y fuera de este lugar.

A todos los que alguna vez me preguntaron, como iba, qué tal, cuándo presentaba, "¿Y tu proyecto?", todos ellos, todos, y han sido muchos, me han animado a terminar, me han dado el último empujón.

Y por último, pero no menos importante, a Fer, por quien me esfuerzo, por quien escogí este proyecto, por su apoyo, por recordarme cuando se me olvidaba cuánto me gusta la informática, y por su incansable espíritu investigador, las horas de biblioteca, las horas de ordenador, las horas de portátil… y por sacarme una sonrisa, aunque me enfadara con él por animarme tantas veces como me he atascado con este proyecto. Me has dado el ánimo para terminar, para empezar lo que viene y a aprender todo lo que me queda por vivir a partir de ahora.

Índice

Capítulo 1. Introducción

1.Introducción

Según la definición de del Diccionario de la Real Academia Española, la Inteligencia Artificial es el "Desarrollo y utilización de ordenadores con los que se intenta reproducir los procesos de la inteligencia humana."

Desde tiempos inmemoriales el hombre siempre ha intentado superarse a sí mismo y alcanzar nuevos retos, plantearse inquietudes.

Hoy día vivimos en un mundo en el que la tecnología y los ordenadores se nos han hecho un compañero imprescindible en nuestra vida diaria; hablamos por mensajería instantánea, se hacen amigos por internet y se consultan aparatos electrónicos para hacer una receta en la cocina.

En el mundo de los juegos ha ocurrido de igual manera, el hombre ha tenido sus inquietudes en este ámbito. El primer juego de ordenador, ***"Lanzamiento de Misiles",*** presentado en 1947 por Thomas T. Goldsmith y Estle Ray Mann, no se puede considerar un videojuego, porque no existía movimiento de video en la pantalla, pero sí se convirtió en el primer experimento real con un dispositivo electrónico de simulación.

Años después en 1952, Alexander Sandy Douglas presenta su tesis de doctorado en matemáticas en la Universidad de Cambridge (Inglaterra) sobre la interactividad entre seres humanos y computadoras, que incluía el código del Tic, Tac, Toe, el primer juego gráfico computerizado.

En 1958 William Higinbotham programa "Tennis for Two", el primer juego con videoanimación y que permitía la interacción entre dos jugadores. Se visualizaba en un osciloscopio de laboratorio conectado a un computador, donde se mostraban dos líneas perpendiculares que simulaban el suelo y una red de pista de tenis.

Más adelante nació SpaceWar, considerado el embrión de los videojuegos galácticos. Fue programado por Steve Russell en 1961, en una computadora PDP-1 en el MIT

Podrían llenarse miles de hojas con juegos que los seres humanos han programado para probar su inteligencia.

Los juegos de mesa siempre ocuparán lugar en nuestro ocio, pero con la evolución de la tecnología no es descabellado pensar que aquellos juegos que no se han llevado al formato interactivo puedan caer en el olvido.

1.Introducción

La elección de Camelot para la creación de un programa de ordenador con una Inteligencia Artificial que presentara un juego competitivo, no es más que el ánimo de rescatar los clásicos del pasado para darles una oportunidad en este mundo en constante evolución; con la intención de conseguir un juego, cuyo creador sea incapaz de derrotar. En el documento a continuación se presenta su análisis, desarrollo y resultados obtenidos del programa con el nombre de BOU (transcripción fonética de reverencia en inglés)

Con el ritmo de avance que posee la tecnología y la capacidad que posee el ser humano de computerizar su pensamiento abstracto no es complicado imaginarse un futuro en el que las máquinas superan la inteligencia humana.

Capítulo 2. Estado de la cuestión

En este capítulo se tratará la situación actual que rodea al proyecto, estudios realizados al respecto, y detalles a tener en cuenta para su desarrollo.

2.1. El juego de Camelot

Camelot es un juego de estrategia que posee 129 años de historia. Es bastante fácil de aprender y, en general, para los principiantes, muy divertido de jugar. George S. Parker fue su creador, y su intención era inventar una actividad de recreo no tan difícil como el ajedrez, pero considerablemente más variado que las Damas. En los siguientes apartados se aprenderá un poco más acerca del nacimiento de este juego.

2.1.1. Historia de Camelot

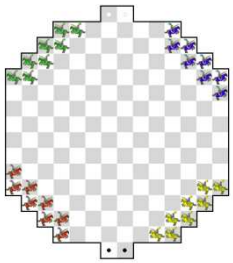
Uno de los primeros juegos publicados por Parker Brothers, Camelot fue inventado a finales del siglo XIX (sobre 1882) por Swinnerton George Parker y publicado originalmente bajo el nombre de Chivalry (caballería**)**. Fue reeditado en 1930 bajo el nombre de Camelot y desde entonces, hasta 1960 (en la que está fechada su última edición), han surgido numerosas variantes y decenas de ediciones. Pero aún estando descatalogado, todavía se mantiene un pequeño núcleo de seguidores acérrimos que esperan un posterior renacimiento

De la idea de George Parker, surgió un juego excepcionalmente táctico casi desde el primer paso, por lo que se puede llegar rápido a la conclusión de la partida., mezcla de Halma (damas chinas) y American Checkers (Damas). Cuando finalmente fue publicado por Geo. S. Parker & Co en 1887 (y posteriormente por Parker Brothers en 1888), supuso las delicias de los grandes expertos en Ajedrez y Damas, pero no ganó popularidad rápidamente con el público en general, a pesar de que Parker lo denominó "el mejor juego en 2000 años"

Desde que lo inventó en 1882, Parker nunca perdió su entusiasmo por el juego, por ello en 1930 hizo algunos cambios, para que Parker Brothers lo reeditara bajo el nombre de Camelot. Se siguieron realizando cambios en las normas durante 1931. Disfrutó de su mayor popularidad en ésta década.
Se sacaron a la venta más de 50 diferentes ediciones de Camelot, incluyendo una edición con un sello de oro estampado sobre cuero y una edición de caoba. Hubo ediciones de torneo, ediciones periódicas, y ediciones de bajo coste.

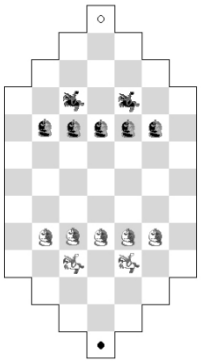
De todas las variantes que surgieron durante su época de auge, Parker Brothers ha comercializado algunas de ellas. Por ejemplo, Point Camelot, variante de la jugada

en torneos en la que se cuentan y anotan puntos fue editada en 1931; Grand Camelot, de cuatro jugadores en un tablero con mayores dimensiones, Three-handed Camelot (*Camelot* para tres jugadores), Four-handed Camelot *(Camelot* para cuatro jugadores), fueron lanzadas en 1932. Cam, una variante que se juega en un tablero en miniatura, salió en 1949. Surgieron más versiones, Camelotta, Camette, Anti-Camelot, Cam-3D, Tri_camelot, Grand Cam, y Grand Camette pero ninguna de estas variantes alcanzó nunca la popularidad del juego básico y ya apenas existe información conocida. En la figura 2.1 se muestran imágenes de los tableros de las distintas versiones de Camelot.

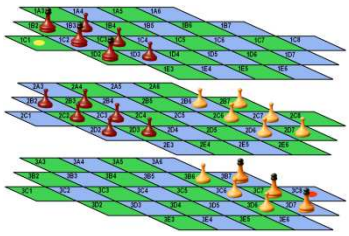


Four-handed Chivalry

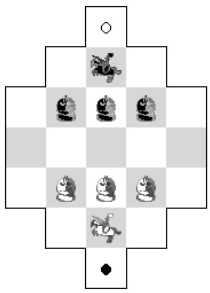
Grand Camelot



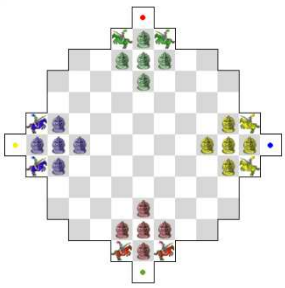
Cam



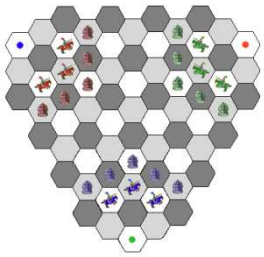
Cam-3D



Camette



Grand Cam



Tri-Camelot

Figura 2. 1

Camelot fue finalmente descatalogado en 1968, luego reeditado como Inside Moves en 1985, para ser descatalogado nuevamente en 1986. Debido a su antigua popularidad, todavía es posible obtener alguna copia a través de eBay o similares. También se puede reproducir en un PC, utilizando programas como Zillions games.

Entre los jugadores de Camelot hay personajes tan ilustres como José Raúl Capablanca, campeón mundial de ajedrez de 1921 a 1927, y Frank Marshall, EE.UU. Campeón de Ajedrez de 1907 a 1936. Sidney Lenz y Milton Work, dos jugadores de bridge de fama mundial, también jugaban al juego.

La Federación Mundial de Camelot (WCF), es una organización sin ánimo de lucro, sucesora de los clubes asociados Camelot de América y dirigida por Michael Nolan Wortley.

Se formó en 1999 en un intento de preservar y difundir el juego. El WCF ha introducido algunas aclaraciones a reglas, adiciones y cambios. La Web de la Federación se creó en el año 2000 y posee subscripción gratuita.

2.1.2. Reglas del juego de *Camelot*

Camelot se juega sobre un tablero rectangular de 160 casillas (12x16), retirando tres casillas de cada una de las cuatro esquinas, y añadiendo cuatro casillas fuera del rectángulo principal, centradas dos en la parte superior y dos en la parte inferior de la tabla. Estos dos cuadrados se denominan zonas de los Castillos (Figura 2.2). Cada jugador comienza con catorce piezas, cuatro Caballeros y diez Hombres.

El **objetivo del juego** es ser el primer jugador en ocupar el castillo del oponente (los dos cuadrados marcados) con dos de sus propias piezas, o en capturar todas las piezas de tu oponente, conservando al mismo tiempo dos o más de sus propias piezas.

Caballeros y hombres pueden avanzar por el tablero de tres maneras, ya sea en ortogonal como en diagonal: *Plain move*, *canter* y *Jump*. Las piezas de hombre pueden hacer cualquiera de estos tres movimientos, pero sólo un tipo de movimiento por turno. Las piezas de Caballero tienen una cuarta opción, la combinación llamada *Knight's Charge* (la carga del caballero). Se explican en detalle estos movimientos a continuación.

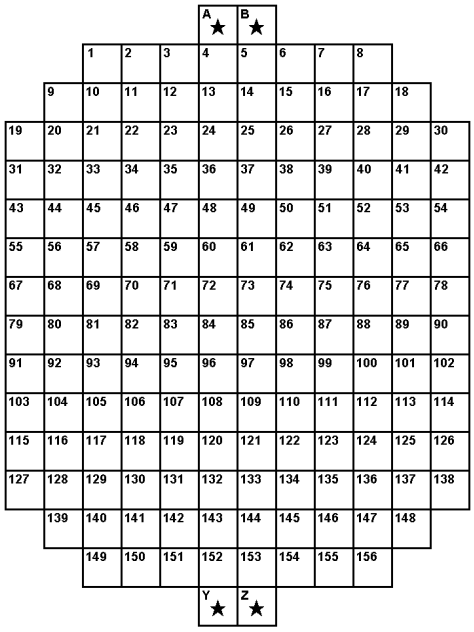


Figura 2. 2

Plain Move (*movimiento simple*): Una pieza (ya sea Hombre o Caballero) puede desplazarse un cuadrado en cualquier dirección (horizontal, vertical o diagonalmente) a cualquiera de las casillas libres adyacentes.

Canter (*Galope*): Una pieza (ya sea Hombre o Caballero) puede saltar en cualquier dirección (horizontal, vertical o diagonalmente) sobre una pieza de su mismo color (ya sea hombre o Caballero) que esté ocupando una plaza adyacente, siempre que exista una plaza desocupada contigua a la adyacente en la línea directa en la que se realice el *galope*. Las piezas sobre las que se ha hecho *galope* no se retiran del tablero.

Un jugador puede "galopar" sobre más de una pieza durante el mismo movimiento, pero no puede hacer una "*galope* circular", es decir, que termine en la misma casilla desde la que comenzó.

En caso de realizar el *galope* sobre más de una pieza en un movimiento, la dirección de éste puede ser variada después de cada salto.

Si el *galope* de una pieza de Caballero le lleva junto a una pieza enemiga que puede capturar está obligada a hacerlo mediante *The Knight's Charge* (se explica a continuación), a no ser que tomando otra dirección, pueda capturar una o más piezas enemigas en otros lugares. Si el jugador decidiera finalizar el movimiento, sin realizar la captura, el oponente puede forzarlo a llevarla a cabo, o bien, dejar la pieza tal y como estaba, pasando el turno, a su elección.

Jump (*salto*): Una pieza (ya sea Hombre o Caballero) puede saltar en cualquier dirección (horizontal, vertical o diagonalmente) sobre una casilla contigua ocupada por un adversario (ya sea hombre o Caballero), siempre que exista una plaza desocupada inmediatamente más allá en línea recta al *salto* que puede hacerse. La pieza sobre la que se ha saltado en este movimiento es capturada y eliminada inmediatamente de la partida.

Un jugador puede saltar más de una pieza oponente en el mismo movimiento y la dirección de éste puede variar después de cada *salto*.

El movimiento de salto está obligado siempre y cuando sea posible realizarlo; solamente si hay más de una forma por la cual una pieza enemiga puede ser capturada, el jugador que realiza el *salto,* podrá tomar su elección.

Al igual que con el movimiento de galope, si un jugador que está en condiciones de saltar sobre una pieza del rival, hace un movimiento con el cual no se captura ninguna pieza o bien finaliza una cadena de saltos pudiendo continuar capturando, el oponente puede forzarle a hacer una captura o a dejar la pieza tal y como está pasando turno, como prefiera.

The Knight's Charge (*la carga del caballero*): Un Caballero (sólo, los Hombres no pueden) puede combinar el *galope* y el *salto* en un solo movimiento.

Se comienza con un *galope* en cualquier dirección (horizontal, vertical o diagonalmente) sobre una o más piezas de tu color (ya sea Hombre o Caballero), variando la dirección del *galope* si se desea, para llegar a una casilla junto a una pieza enemiga sobre que la sea posible saltar (Hombre o Caballero) y capturar para eliminarla del tablero, todo formando parte del mismo movimiento. Una vez realizado el salto, el jugador está obligado a seguir capturando piezas enemigas, si le es posible, y puede ser

en cualquier dirección. Este movimiento debe ese orden de movimiento: Comenzar con el *galope* (las veces que sea necesario) y terminar con el *salto*(los que sean posibles).

Un Caballero no está obligado a hacer *la carga del caballero*, pero si tras llevar a cabo un *galope* se finaliza junto a una pieza enemiga que puede capturar, sí debe hacerlo, a menos que por otra vía en ese mismo movimiento pueda capturar una o más piezas enemigas en otros lugares. Si pudiendo capturar tras un *galope*, el jugador finaliza su movimiento sin realizar ningún *salto* pudiendo hacerlo, el oponente puede forzar al jugador a hacer una captura, o pasar el turno, según prefiera.

Vemos en la figura 2.3 un ejemplo de movimiento de carga del caballero:

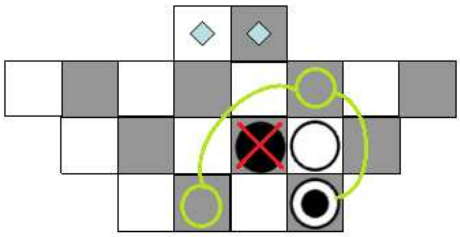


Figura 2.3

Los 24 posibles movimientos

Tomada una pieza cualquiera del tablero, se pueden realizar como máximo 24 posibles movimientos diferentes (inicialmente, descartando los movimientos encadenados), dependiendo de su situación en el tablero y las piezas colindantes. En realidad, hay 24 posibles combinaciones, pero cada pieza sólo puede realizar como mucho 8 movimientos simples, porque el hecho de poder realizar alguno de ellos descarta poder llevar a cabo el resto; cómo se explica a continuación en la figura 2.4

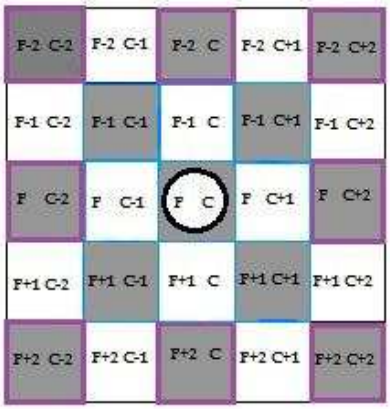


Figura 2.4

Hay 8 posibles movimientos simples en los que la casilla avanza hacia una contigua siempre que ésta no esté ocupada por otra pieza. Se indican las ocho posibilidades en la figura 2.5.

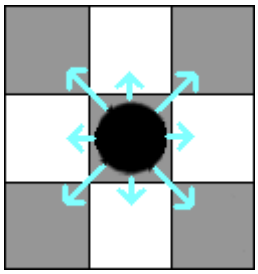


Figura 2.5

Existen 8 posibles movimientos de *galope* (saltar sobre una pieza de tu mismo color), en los que la casilla contigua debe estar ocupada por una pieza amiga y se debe comprobar que la casilla destino al realizar el salto sobre ella siguiendo la misma dirección, está vacía. La pieza sobre la que se ha saltado, no desaparece del tablero. Se pueden observar tres ejemplos de *galope* en la figura 4.27.

Y por último, existen 8 posibles movimientos de *salto* (captura de una pieza oponente), en los que la casilla contigua, como en el caso anterior, debe estar ocupada por una pieza enemiga, así como vacía la casilla destino al pasar por encima de ella siguiendo la misma dirección. Esta pieza sobre la que se ha saltado si desaparece del tablero, como se observa en los 5 ejemplos de captura de la figura 2.6.

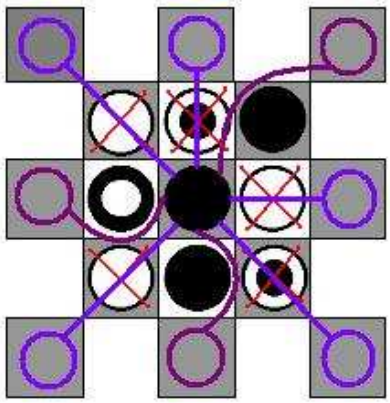


Figura 2.6

Abandono de la partida:

Se llega a este punto si:

- Ambos jugadores así lo acuerdan.
- Ninguno de los jugadores está en condiciones de realizar un movimiento legal.
- A ambos jugadores no les queda más que una pieza en el tablero.
- Se reclama por cualquiera de los jugadores si en el transcurso de la partida se han repetido los movimientos de cada jugador en los últimos tres turnos
- Se reclama por cualquiera de los dos jugadores cuando en los 50 últimos movimientos consecutivos de la partida por cada una de las partes, han sido sin ninguna captura ni movimientos en las casillas del castillo oponente.

Las casillas del castillo:

Un jugador, teóricamente, no puede mover ninguna de sus piezas (Caballero u Hombre) en las casillas de su propio castillo; pero si una pieza enemiga llega a una casilla contigua a las de su castillo, el jugador puede realizar un *salto*, o llevar a cabo *la carga del caballero*, sobre esa pieza del rival entrando en las casillas de su castillo. Ese movimiento debe terminar allí, aún si hay oportunidad de continuar el *salto*. Esta pieza está obligada a abandonar su castillo en el siguiente turno, sin excepción. Si existe una oportunidad de capturar una pieza rival mediante un *salto* o *la carga del caballero* a la hora de salir, tiene la obligación de hacerlo, en lugar de abandonar el castillo mediante el *galope* o un *movimiento simple*

Cuando una pieza ha entrado en el castillo de su oponente, no puede salir de él, pero se le permite realizar movimientos entre esas dos casillas (denominado cómo "*movimiento del castillo*") pero no más de dos veces por juego.

2.1.3. El estado actual de la competición de Camelot

Desde 1921 se llevaron a cabo campeonatos mundiales del juego de Camelot en los que participaban grandes jugadores. El vencedor pionero en estos torneos concertados desde su primer año de celebración en 1894 fue Enmanuel Lasker, que permaneció en su liderazgo hasta 1920. Jose Raul Capablanca fue el ganador

consecutivo de 1921 a 1927, cuyo título fue arrebatado por Alexander Alexandrovich Alekhine, que venció en 1927 hasta 1935 y después de 1937 a 1946.

Se han retomado recientemente estos campeonatos, gracias a que la WCF organizó un Campeonato del Mundo con doce participantes, que comenzó el 10 de mayo del 2002 y finalizó el 22 de junio de 2003, con la siguiente conclusión: Dan Troyka de Saline, Michigan, fue el ganador del título de Campeón del Mundo de Camelot. Los resultados de las clasificaciones de este torneo fueron las siguientes:

GROUP 1	**DT**	**MT**	**KH**	**Total**
Dan Troyka	x	1	2	**3**
Mark Thompson	1	x	0	**1**
Kerry Handscomb	0	2	x	**2**
GROUP 2	**John**	**MN**	**Jerry**	**Total**
John Lawson	x	0	2	**2**
Michael Nolan	2	x	2	**4**
Jerry LaSala	0	0	x	**0**
GROUP 3	**AP**	**DH**	**JO**	**Total**
Andrew Perkis	x	2	2	**4**
Dan Hale	0	x	0	**0**
Jim Oliver	0	2	x	**2**
GROUP 4	**DH**	**PY**	**ZI**	**Total**
Don Haffner	x	0	0	**0**
Paul Yearout	2	x	2	**4**
Zillions AI	2	0	x	**2**

Tabla 2.1

SEMI-FINAL 1	AP
Dan Troyka	2
Andrew Perkis	1
SEMI-FINAL 2	**AP**
Michael Nolan	2
Paul Yearout	0

Tabla 2.2

WORLD CHAMPIONSHIP	AP
Dan Troyka	3
Michael Nolan	1

Tabla 2.3

Un nuevo Campeonato Mundial se llevó a cabo en 2008, comenzando el 20 Julio, pero por el momento sólo se ha llevado a cabo la primera ronda de eliminación y el resultado de este torneo está aún por determinar.

Varios de los miembros de la WCF se involucraron en la creación de un programa de ordenador con el que jugar a Camelot vía Internet que ofreciera tanto la opción de enfrentar a dos jugadores humanos como una opción de Inteligencia Artificial que permite jugar entre un humano y un ordenador. Las pruebas, normas, estrategias y tácticas fueron revisadas por los miembros de WFC en Canadá y los EE.UU. Un miembro de la Federación Inglesa llamado Rick Fadden consiguió obtener el primer programa de PC que cumplía con éxito las expectativas. Se denominó CHAXX. La versión actual CHAXX 1.13 es lo suficientemente eficaz como para derrotar a la mayoría de rivales humanos. Se sigue trabajando sobre él, añadiéndole mejoras.

Entre el 14 de septiembre y el 18 de septiembre del 2008 se llevó a cabo un campeonato mundial de computadores de Camelot, entre Zillions Camelot y Chaxx 1.13; torneo que finalizó rápidamente, situando como vencedor a Chaxx 1.13.

Ha habido una serie de intentos de programar nuevas simulaciones del juego de Camelot, pero han resultado fallidos. La competición mundial del juego de Camelot por ordenador está abierta a quien quiera presentar sus proyectos, pero se exigen una serie de requisitos a la hora de programar una versión: Un interfaz que represente el juego, (las piezas, el tablero, los movimientos…), fidelidad con las reglas oficiales del juego (adjuntan una lista de reglas obligatorias), y un algoritmo que evalúe las posiciones y los posibles movimientos, durante el juego (Para más detalles consultar el apartado 2.6.1 de este capítulo).

Cómo no surgen rivales a la altura del programa campeón mundial de Camelot CHAXX, se establecen retos a pequeña escala para mantener la motivación. Lo más reciente, este 2011, la federación mundial de Camelot, prevé organizar un evento que enfrente al humano contra la maquina: Campeonato mundial de Camelot entre CHAXX y Dan Troyka, estableciendo un tiempo límite de 2h y media o un máximo de 40 movimientos. De este modo, cada cierto periodo de tiempo, se planean reuniones para continuar con la tradición de este entretenido juego, que posee ya casi 130 años de antigüedad.

2.2. Clasificación de los juegos:

Existen numerosas formas de clasificar un juego, dependiendo del criterio con el que se defina. A continuación se especifican una serie de criterios con sus correspondientes tipos de juegos:

- Suma nula: Son aquellos juegos en los que el éxito final de un jugador implica la pérdida del resto de jugadores.
- Información completa / no completa: Es de información completa si todos los elementos de decisión que precisa uno de los jugadores para estudiar la siguiente acción están disponibles (Ajedrez, Damas); siendo no completa en caso de que no se precise de esta información (póker, Cluedo)

- Aleatorios: Dependiendo si interviene o no el azar. De esta manera, estos juegos en los que interviene el azar son evidentemente de información no completa (como hemos indicado antes, el póker).
- Cooperativos/ no cooperativos: Si algunos jugadores pueden aunar sus esfuerzos en la consecución de un fin común (Trivial Pursuit, tute por parejas, Monopoly,…) el juego es cooperativo. En caso contrario, se trata de un juego no cooperativo (Mentiroso, Uno)
- Número de jugadores o agentes: por ejemplo de un solo jugador(solitario, Mah-jong), de dos jugadores (backgammon) o multijugador, (Pocha)
- Dinámicos: si el fin perseguido por el resto de los oponentes resulta perfectamente conocido a cada uno de los jugadores (Othello, Hexagon) el juego es dinámico. En el caso contrario, no lo es (Risk, Carcassone).

El juego de Camelot, descartando las versiones no oficiales que se han presentado en el capítulo anterior (multijugador, por parejas...), se ajusta a la descripción de un juego de suma nula, de información completa, no aleatorio, no cooperativo, en el que intervienen dos agentes y de carácter dinámico.

2.3. Algoritmos de búsqueda:

Hay una gran variedad de algoritmos de búsqueda, y en cada uno de ellos se ha invertido un gran esfuerzo para producir mejoras y poder obtener finalmente, una búsqueda más eficiente dentro del desarrollo de juegos. A continuación se muestra una breve introducción a aquellos algoritmos de búsqueda más importantes con sus características básicas y más representativas y una descripción de aquellos utilizados en el desarrollo de este juego.

2.3.1. Conceptos generales:

La figura a continuación (Figura 2.7), muestra los términos más usuales empleados en los algoritmos de búsqueda.

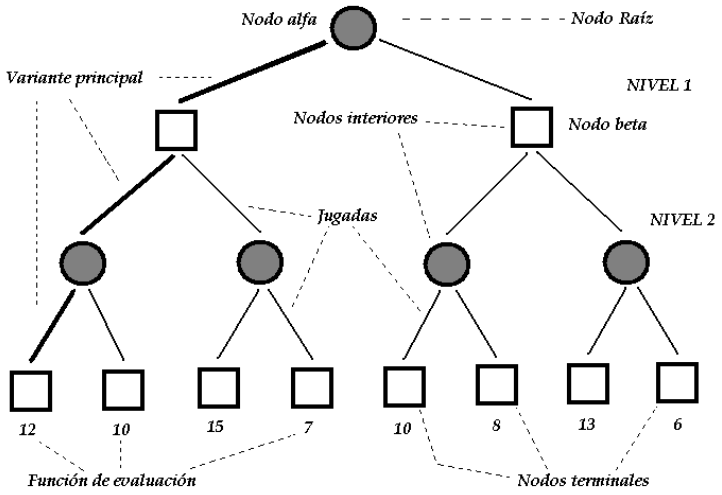


Figura 2. 7

A partir de una posición o tablero inicial conocido, denominado **nodo inicial** o **raíz**, se desarrolla un árbol con todas las combinaciones de tablero posibles para ambos jugadores hasta alcanzar un **nivel de profundidad** d. El árbol generado recibe el nombre de árbol de búsqueda.

Se denominan **nodos terminales** aquellos que no tienen descendientes, bien porque el tablero que representan no puede progresar a otras posiciones, o bien porque están a la profundidad de búsqueda prefijada. Entre el nodo raíz y los nodos terminales están todos los **nodos intermedios** o **internos**, también llamados no terminales. Dichos nodos se dividen en:

- **Nodos alfa:** Representados con un círculo, corresponden a aquellas jugadas en las que el ordenador toma la decisión.
- **Nodos beta:** Representadas con un cuadrado, son aquellos en los que corresponde tomar una decisión al usuario oponente.

Habitualmente, tal y como se muestra en la anterior figura, es los nodos pares hay, exclusivamente, nodos alfa (el nodo raíz está a profundidad 0) y, en los impares, nodos beta.

Además, es necesario disponer de una función para la valoración de la calidad de cualquier tablero o nodo. Dicha función se denomina **función de evaluación** y se debe aplicar, al menos, a los nodos terminales.

En la figura a continuación (figura 2.8) se ilustra el concepto de **descendientes de un nodo** cualquiera.

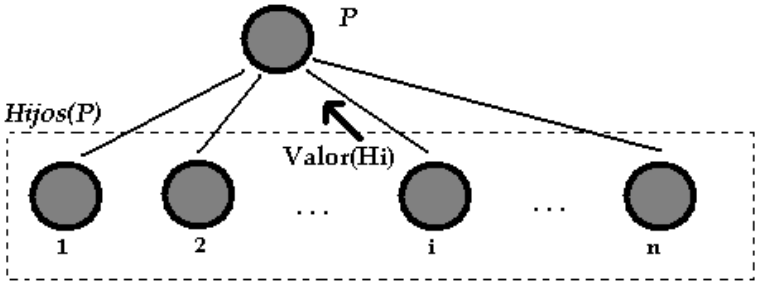


Figura 2. 8

Si un nodo puede progresar a otras situaciones, a estás se les llama hijos, y a él se le llama padre. Si P es el nodo padre, sus hijos se identificarán con el conjunto Hijos (P). El hijo i-ésimo del nodo P se denota como H_i. Además cada nodo, después de su generación, devuelve un valor a su padre que es una evaluación de las posibilidades que hay a partir de él, si es intermedio, o el valor de su función de evaluación si es terminal. Al valor devuelto por el hijo i-ésimo del nodo P se le nota como valor (H_i) que es, en cualquier caso, el valor del nodo H_i.

A partir del árbol de búsqueda, manipulando de diferentes maneras los conceptos que se han expuesto, se pueden aplicar cualquiera de los algoritmos de búsqueda que se presentan en las siguientes secciones.

De la aplicación de los algoritmos de búsqueda a un nodo inicial se obtiene información sobre qué alternativa de las posibles es la mejor. Dicha información consiste típicamente, en la elección de una única alternativa y una valoración se su validez.

De hecho, la variante principal y el valor definitivo del árbol de búsqueda están íntimamente relacionados puesto que dicho valor es el del nodo terminal de la variante principal.

2.3.2. Procedimientos de búsqueda recursiva:

Los ordenadores son, en realidad, grandes dispositivos para el cálculo de enormes volúmenes de información, es inmediato entender que los primeros algoritmos

de búsqueda confiaran en esa capacidad bruta y consistieran en implantaciones correctas de la lógica que debe seguirse en el proceso de búsqueda. Esto es, en encontrar aquellas combinaciones que maximizaran la posibilidad de ganar para un jugador, al mismo tiempo que se minimizan las de su oponente. Sin embargo, como se verá, pronto se descubrieron otros mecanismos para desestimar aquellas líneas que, con seguridad, nunca pudiesen formar parte de la solución buscada.

2.3.2.1. Minimax:

El algoritmo de búsqueda Minimax es la primera aportación formal a la teoría de juegos. En realidad, el resto de los algoritmos se basan en éste o en alguno de sus fundamentos.

Ideado por Von Neumann (1928), consiste simplemente en establecer el compromiso para las mejores jugadas de ambos contendientes. Esto es, en una visión realista no se deben esperar situaciones favorables siempre que el oponente pueda evitarlo, ni tampoco desfavorables, si el jugador puede rehuirlas.

El procedimiento es el siguiente:

- Si un nodo P es terminal, su valor es el de la función de evaluación para el jugador para el que se resuelve el árbol de búsqueda.
- En otro caso su valor será:
    - El máximo de los valores de sus hijos si es un nodo alfa(corresponde a que el ordenador escoge aquel valor que más le beneficia)

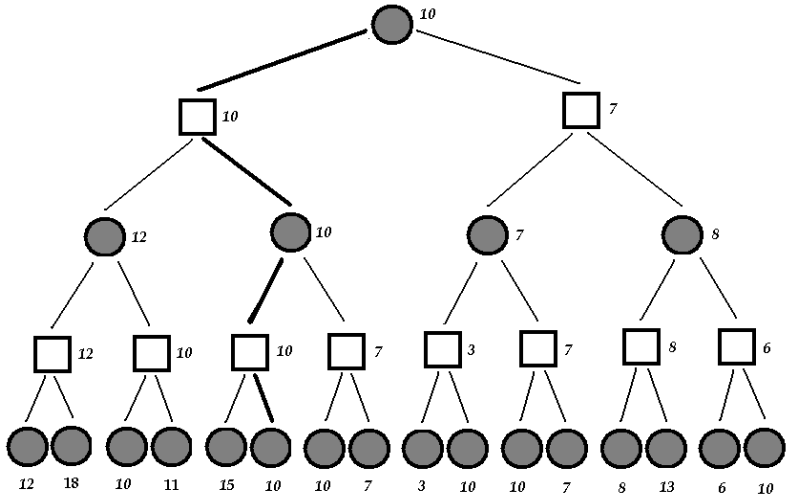
$$Valor(P)=\max_{i=1,n}\{valor\ (H_i),\ \forall\ H_i \in Hijos(P)\}$$

    - El mínimo de los valores de sus hijos si es un nodo beta(el ordenador escoge el valor en el que su oponente maximiza su opción y por tanto, minimiza la suya)

$$Valor(P)=\min_{i=1,n}\{valor(H_i), \forall H_i \in Hijos(P)\}$$

- El algoritmo finaliza cuando se dispone de la puntuación del nodo inicial. La mejor jugada será la que devolvió el valor elegido por el nodo raíz.

A continuación se muestra un ejemplo de aplicación de este algoritmo (Figura 2.9). Junto a cada nodo, no terminal, se representa su valor Minimax. Se puede observar que la variante principal se caracteriza por asignar a todos los nodos que recorre, el valor Minimax o el valor del nodo terminal, según corresponda



.

Figura 2. 9

En el primer apartado del capítulo 8.Apéndices, se encuentran las normas para la escritura de pseudocódigos que se ofrecen en los siguientes apartados. A continuación, en el apartado 8.2 se incluye el código de Minimax utilizado para el desarrollo del juego de Camelot. Cómo se ve, la búsqueda se realiza recursivamente (como en el resto del algoritmos de búsqueda) y, por lo tanto, primero se referencia al hijo más a la izquierda y se progresa hacia la derecha. Esta observación distingue al hijo más a la izquierda.

2.3.2.2. Negamax

Del Minimax puede obtenerse otro algoritmo conocido como Negamax, advirtiendo (Baudet, 1978) que:

$Max\{min\{x_1,x_2,\ldots\},min\{y_1,y_2,\ldots\},\ldots\}=max\{-max\{-x_1,-x_2,\ldots\},-max\{y_1,y_2,\ldots\},\ldots\}$

Por lo que las reglas anteriores para la asignación de valores a nodos no terminales queda ahora como se muestra a continuación:

$$Valor\ (P)=max_{\ i=1,n}\{-valor\ (H_i),\ \forall\ H_i\ \in Hijos(P)\}$$

Si se aplica la función de evaluación en los nodos terminales para el jugador al que le toca mover en ellos y no, exclusivamente, para el que se está resolviendo el árbol de búsqueda.

A continuación se muestra un ejemplo aplicado de árbol de búsqueda del algoritmo Negamax (Figura 2.10):

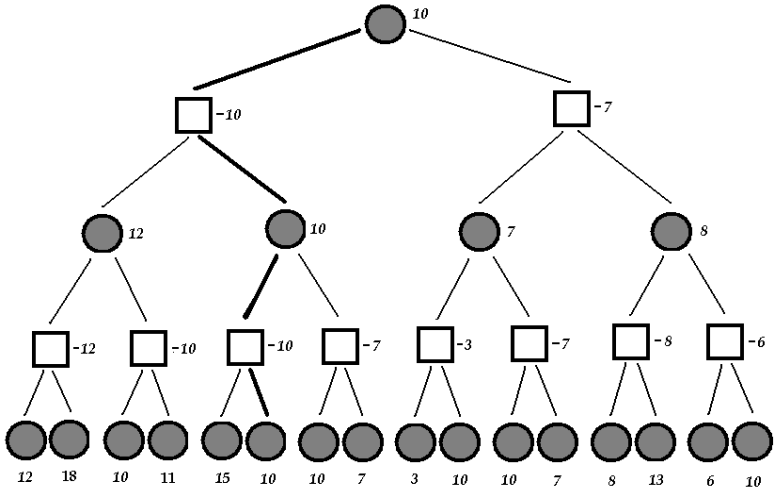


Figura 2. 10

En realidad, esta variante del Minimax no aporta velocidad, sino elegancia. En la figura anterior se ha mostrado el mismo árbol representado antes con el algoritmo Minimax. Se puede apreciar que los nodos de la variante principal cambian su signo en cada nivel. Con este algoritmo, las puntuaciones de los nodos terminales son las mismas que en el ejemplo Minimax debido a que el nivel de búsqueda ha sido par (d=4). De

ahora en adelante, y mientras no se indique lo contrario se utilizará el esquema Negamax.

No se muestra en el capítulo de apéndices, el código del Negamax por no ser uno de los algoritmos que se han desarrollado para el proyecto; al no aportar ventaja ninguna respecto del Minimax, se descartó como opción de estudio. En cambio, se incluye el pseudocódigo en el apartado 8.1.2 por la relevancia informativa que posee.

2.3.2.3. Alfa-beta

No se sabe con certeza quien desarrolló el algoritmo de búsqueda de Alfa-beta, y sin embargo, constituye un gran avance con respecto a los algoritmos anteriores al introducir nuevos conceptos en la búsqueda por ordenador.

Para su aplicación, se mantienen dos valores, el valor alfa, o valor optimista, y el valor beta, o valor pesimista. Dichos valores se mantienen en el intervalo (α, β), durante todo el proceso de búsqueda. A este intervalo se le conoce como **ventana de búsqueda**, o simplemente ventana. Los valores de α y β de un nodo resultan, no solo de las exploraciones que se realicen a partir de él, sino de las que se hicieron en los nodos que le preceden. Por ello, es inmediato comprender que si el valor más optimista (α) es mayor o igual que el valor más pesimista (β) dicho nodo no afectará al valor final del algoritmo de búsqueda y nunca formará parte de la variante principal. En tal caso, se practica una poda.

Para una mejor comprensión de las diferencias entre el Minimax y el Negamax, se explicará este algoritmo sobre las dos arquitecturas de búsqueda.

El algoritmo Alfa-beta sobre el Minimax queda, de la siguiente manera:

- Inicialmente, α y β toman los valores -∞ y +∞ respectivamente.
- Si un nodo P es terminal, su valor es el de la función de evaluación para el jugador al que le toca mover en dicho nodo. Por convención se toma:

$$\alpha = \text{valor}(P)$$

$$\beta = \text{valor}(P)$$

y no se realiza ninguna comprobación de poda.

- En otro caso su valor será:

$$\text{Valor}(P) = \max_{i=1,n}\{\text{valor}(H_i), \forall H_i \in \text{Hijos}(P)\}$$

Y:

$$\alpha = \max\{\alpha, \text{valor}(P)\}$$

si el nodo P es un nodo alfa. Si el nodo P es un nodo beta:

$$\text{Valor}(P) = \min_{i=1,n}\{\text{valor}(H_i), \forall H_i \in \text{Hijos}(P)\}$$

Y:

$$\beta = \min\{\beta, \text{valor}(P)\}$$

- Además, en cada nodo se comprueba la condición de poda:

$$\alpha \geq \beta$$

y si esta se cumple, no se exploran más hijos y se devuelve del valor almacenado hasta el momento

- A partir de cada nodo P se realizan llamadas recursivas al algoritmo Alfa-beta con los valores de α y β actuales para obtener su puntuación.
- El algoritmo finaliza cuando se dispone de la puntuación del nodo inicial. La mejor jugada será la que devolvió el valor elegido por el nodo raíz.

El algoritmo Alfa-beta, tomando como referencia el Negamax en esta ocasión, queda, por tanto, como se indica a continuación:

- Inicialmente α y β toman los valores de -∞ y +∞ respectivamente.

- Si un nodo P es terminal, su valor es el de la función de evaluación para el jugador para el que se resuelve el árbol de búsqueda. Por convención se toma:

  α = valor (P)

  β = valor (P)

  y no se realiza ninguna comprobación de poda.

- En otro caso, su valor será:

$$\text{Valor (P)}=\max_{i=1,n}\{-\text{valor }(H_i),\ \forall\ H_i\ \in \text{Hijos(P)}\}$$

  Y:

$$\alpha = \max\{\alpha\ ,\ \text{valor(P)}\}$$

- En cada nodo se comprueba la condición de poda:

$$\alpha\ \geq\ \beta$$

  y si se cumple no se exploran más hijos y se devuelve el valor almacenado hasta el momento.

- A los hijos que se generan a partir de P se les asigna como valor alfa, -β, y como valor beta, -α.

- El algoritmo finaliza cuando se dispone de la puntuación del nodo inicial. La mejor jugada será la que devolvió el valor elegido por el nodo raíz.

En la figura que se muestra a continuación (figura 2.11), vemos un ejemplo de aplicación del algoritmo Alfa-beta sobre el Negamax.

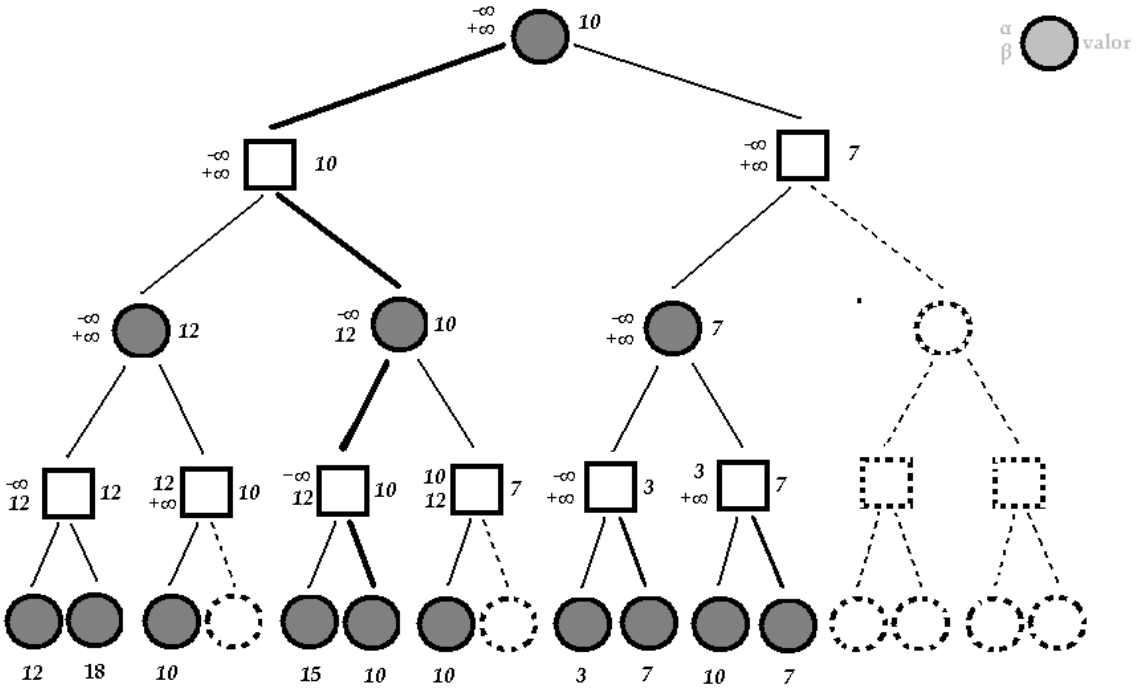


Figura 2. 11

Se puede observar que se han realizado varias podas (indicadas con líneas discontinuas). Por ejemplo, el hijo derecho del nodo inicial, poda a su vez a su hijo derecho debido a que minimiza, escoge el menor valor de sus hijos; fuera cual fuera el valor de esa rama, siempre será menor de 10, que es el valor del hijo izquierdo del nodo inicial. Ya que, si es por ejemplo 3, el nodo padre pasa a valer 3, que es menor que 10 (hijo izquierdo del nodo inicial), y si es por ejemplo 12, el nodo padre pasa a valer 7, que también es menor que 10.

Gérard M. Baudet (1978) denominó poda superficial (*shallow cut-off*) al primer tipo de poda y poda profunda (*deep cut-off*) al segundo.

Sin embargo, el algoritmo Alfa-beta, padece del llamado "efecto horizonte de Berliner" (1974), que pone en evidencia el hecho de que nunca sabemos nada de lo que hay más allá de los nodos terminales. Así, por ejemplo, un sacrificio de calidad en cualquier juego podría no ser reconocido en el árbol de búsqueda si ocurre en un nodo terminal, puesto que entonces se asigna una puntuación muy baja sin saber que a continuación se puede conseguir una posición muy ventajosa. Solo podría solucionarse con un aumento

de la profundidad de la búsqueda, lo que probablemente supondría un aumento en la ocupación de memoria y un retardo en el tiempo de búsqueda.

Desde que se comenzó a usar el algoritmo Alfa-beta, este ha sufrido muchas modificaciones e innovaciones. Algunas de ellas se describen más detalladamente en el capítulo siguiente **Técnicas de aplicación**, y dan lugar a versiones mejoradas y más eficientes de este algoritmo. En el capítulo de apéndices se puede observar el código de este utilizado en el desarrollo del programa de Camelot en el apartado 8.1.3

2.3.2.4. FAlfa-beta

Campbell y Marsland (*Relative Efficiency of Alpha-beta Implementations*, 1983) advirtieron que, dada una estimación V del valor Alfa-beta de un árbol de búsqueda y una estimación del error se esperaría que el valor autentico S estuviera comprendido en el intervalo (V-e, V+e). Sin embargo, podría suceder cualquiera de los casos:

1. $S \leq V-e$
2. $S \geq V+e$
3. $V-e < S < V+e$

Al primero de ellos se le denomina *fallo inferior* (*failing-low*) y al segundo *fallo superior* (*failing-high*). El tercer caso es aquel en que se encontró el valor verdadero entre los límites establecidos.

Evidentemente, e= ∞, plantea la situación más ambiciosa y forzosamente no ocurrirá ni un *fallo inferior* ni un *fallo superior*. Así es, exactamente, como funciona el Alfa-beta.

Sin embargo, el FAlfa-beta (acrónimo de *fail-soft-alfa-beta*), asigna valores de α y β iniciales al nodo raíz distintos de -∞ y +∞ respectivamente, con lo que las podas aumentarán considerablemente, ya que hemos ajustado los límites. Sin embargo de la aplicación del FAlfa-beta debe asumirse el riesgo de pérdida de información si la estimación de α y β no fue correcta y ocurre alguno de los tipos de fallo. El algoritmo de PAlfabeta, otra versión diferente del original, permite resolver este problema. Por consiguiente, FAlfa-beta recorre los mismos nodos que el Alfa-beta con un rendimiento mejor y con muy poca sobrecarga.

En la imagen a continuación (Figura 2.12) se muestra un ejemplo de aplicación del algoritmo que estamos desarrollando para los valores iniciales de α y β.

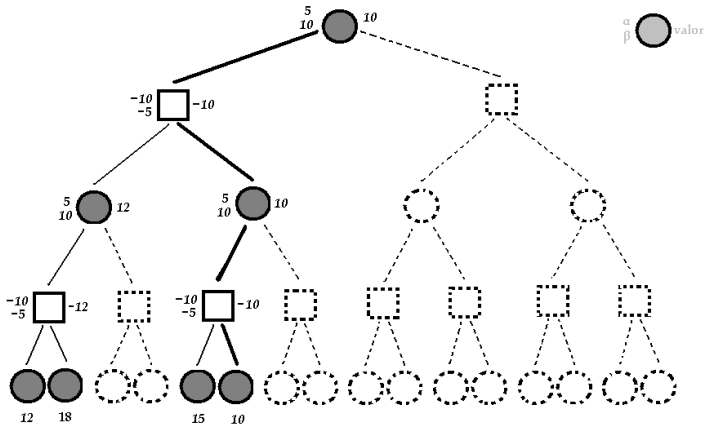


Figura 2. 12

Se observa que se obtiene con el FAlfa-beta la misma variante principal que en el Alfa-beta pero con un ahorro notable de esfuerzo. Sin embargo, si β hubiera sido inferior a 10, el resultado hubiera sido erróneo.

Se puede observar en el capítulo de apéndices en el apartado 8.1.4 el código utilizado en Camelot del algoritmo FAlfa-beta

2.3.3. Técnicas de aplicación

A pesar de que los algoritmos presentados hasta el momento son lo suficientemente eficaces para encontrar la solución al problema de la búsqueda para una situación conocida, es necesario tener en cuenta otros factores, fundamentalmente, el tiempo requerido para hallar la solución, e intentar minimizarlo en la medida de lo posible.

Los resultados pueden mejorarse sustancialmente de dos maneras diferentes:

- Aumentando el número de podas
- Con un mejor aprovechamiento del tiempo disponible.

Para conseguir el primer objetivo se introducen la **ventana mínima**, la **ordenación de nodos** y la **precomputación de límites.**

Para el segundo objetivo se presentan la **profundización iterativa** y **el pensamiento profundo.**

Por último, debe indicarse que todas las técnicas que se exponen a continuación pueden aplicarse a cualquiera de los algoritmos comentados anteriormente.

2.3.3.1. Ventana mínima

La ventana mínima consiste en estrechar el intervalo de búsqueda (α ,β) con la intención de que el número de podas aumente mejorando el tiempo de proceso.

Existen varias técnicas para la reducción de la ventana inicial (-∞, +∞). En ocasiones, se puede aproximar un error que, centrado alrededor de una estimación del valor mínimas de la búsqueda, sirva para construir la ventana mínima (*V*+*e*, *V*-*e*). Lo utiliza el algoritmo FAlfa-beta, como hemos explicado en el apartado anterior.

Sin embargo, con frecuencia, es el diseñador el que implementa la ventana mínima. Al hacerlo sin ningún tipo de control debe asumir una probable pérdida de información aunque utilice modelos correctos de búsqueda. El algoritmo, FAlfa-beta es un ejemplo de ello.

Pero, dejando a un lado estos pequeños aspectos negativos, la ventana mínima sirve para mejorar espectacularmente el tiempo de respuesta. Rosenbloom (1982) estima que, entre otras mejoras, la ventana mínima sirvió para disminuir el tiempo de respuesta en un 66% en su programa de Othello Iago.

2.3.3.2. Ordenación de nodos:

En el algoritmo Negamax, ya se explicó la importancia del hijo más a la izquierda de un árbol de búsqueda o *leftmost-child* y dado que los algoritmos no direccionales, (como PAlfa-beta o Scout, que no se han tratado en este capítulo) no tienen los límites α y β , y en su lugar utilizan, en primera instancia, el valor del primer nodo terminal (*leftmost-child*) que encuentran, es inmediato comprender que si el primer nodo terminal visitado coincidiese con el de la variable principal, el resto del árbol se podaría casi inmediatamente. Esta es la idea que subyace en la ordenación de nodos.

A continuación, se muestra una imagen (figura 2.13) en la que se han ordenado los nodos de todos los niveles del árbol de búsqueda obtenido en el ejemplo del

algoritmo Negamax; posteriormente se ha vuelto a aplicar Alfa-beta (sobre la estructura Negamax):

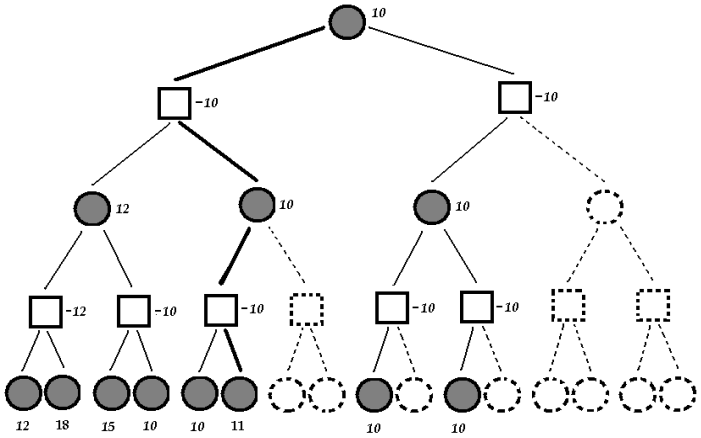


Figura 2. 13

Se puede observar que el número de podas aumenta considerablemente, obteniéndose al final la misma variante principal y la misma puntuación Negamax.

Existen dos formas para realizar la ordenación de nodos:

- Estática: En ocasiones se sabe que una determinada configuración sería preferida a otra. En tal caso, puede mantenerse una tabla ordenada que señale, de manera rígida, qué jugadas deben examinarse primero.
- Dinámica: como en la mayoría de los casos no se sabe cuál es la configuración adecuada para la ordenación, se aplica una función f de ordenación en cada nivel que puede, o no, coincidir con la función de evaluación. El inconveniente que posee esta técnica, es el tiempo extra que requiere la ordenación. Además, este tipo, requiere una ocupación de memoria mayor.

2.3.3.3. Profundización iterativa:

La profundización iterativa (*iterative deepening*) (Reinefeld, 1994)(Korf, 1985) consiste en la exploración del árbol del búsqueda por niveles con el fin de aprovechar mejor el tiempo dedicado a la búsqueda. Así, una vez que se ha establecido este tiempo, se realizan sucesivas llamadas al algoritmo implementado, cada vez a una profundidad mayor, almacenando en cada iteración el último resultado obtenido hasta que el tiempo

se ha consumido, momento en el que devuelve la mejor alternativa encontrada hasta el momento.

Está técnica no es posible llevarla a cabo con un algoritmo recursivo, puesto que para su ejecución se necesita conocer hasta qué profundidad debe llegar. Este planteamiento es como el de una apuesta. Conocido el tiempo que se puede dedicar a la búsqueda, se apuesta a que profundidad llegará, pudiendo ocurrir dos situaciones; podría consumirse el tiempo disponible y no haber acabado aún la búsqueda, en cuyo caso, podría excederse el tiempo global y, al contrario, el algoritmo podría acabar mucho antes de lo previsto, desperdiciándose así un tiempo valioso que hubiera servido para profundizar más.

En el capítulo de apéndices, podemos encontrar el pseudocódigo de esta técnica en el apartado 8.3.1

2.3.3.4. Pensamiento profundo:

El pensamiento profundo (*deep thinking*) utiliza el tiempo dedicado al oponente para pensar la posible respuesta y las contrarrespuestas. Esto es, en realizar una espera activa hasta que se conozca el próximo estado de la partida.

Muchos programas como: Logistello (Buro 1994), actual campeón del mundo de Othello; Bill 3.0 (Lee et al., 1988), campeón del mundo de Othello en 1990; Kitty (Buro, 1994), uno de los mejores programas de dominio público de Othello, etc, implementan esta técnica

2.4. La interfaz gráfica

En este apartado se detallan características acerca de las interfaces gráficas de los programas ya existentes de Camelot.

2.4.1. Creación de entornos gráficos

Existen aplicaciones que, de manera online, te permiten crear un entorno gráfico con las características que desees para poder jugar en un tablero de tres dimensiones. Algunos ejemplos de ellos son: Cyberboard, Thoth y Zuntzu.

Se muestran a continuación tres imágenes del juego de Camelot, procedentes de cada una de estas aplicaciones: Cyberboard (izquierda Figura 2.14), Thoth (derecha Figura 2.14) y Zuntzu (Figura 2.15)

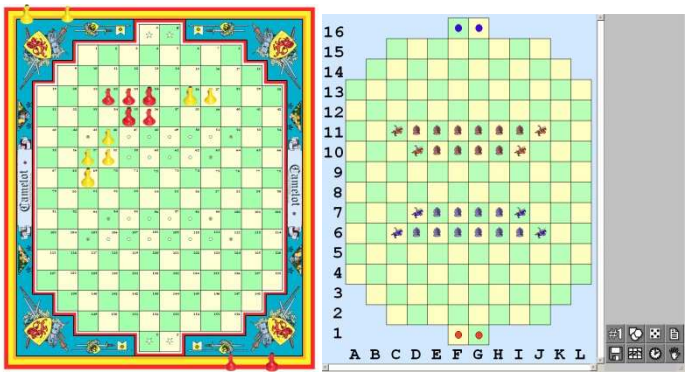


Figura 2. 14

Este tipo de aplicaciones no proporcionan una inteligencia artificial, simplemente el medio gráfico, para que lleves a cabo el juego entre dos usuarios.

Para poder jugar contra el ordenador, es necesario introducir un paquete con una serie de normas, en las que se incluyen las pautas para que el ordenador tome las decisiones pertinentes.

Hoy día se proporcionan a través de la web muchas facilidades para crear juegos con un entorno gráfico, para que cualquier usuario con un nivel básico de conocimientos, pueda crear una aplicación que responda de manera eficaz.

Figura 2. 15

Las plataformas anteriormente nombradas, no son las únicas existentes, hay un gran número de ellas en la red, y algunas de ellas, también poseen una versión programada de Camelot (Por ejemplo, Game Center, BrainKing, GoldToken…).

2.5. Programas ya existentes de Camelot

A continuación en los siguientes apartados se incluye información acerca de los programas ya creados del juego de Camelot.

2.5.1. ZILLIONS, plataforma online de juegos

En la Figura 2.16 se muestra el aspecto de la primera computerizacion libre de errores, que siguen las reglas jugables de los juegos de la familia de Camelot, Camelot, Gran Camelot, Cam y Camette. Fueron escritas por Dan Troyka para la plataforma Zillions (Zillions es una plataforma de pago).

Esta imagen además incluye Chivalry, una nueva variante, que aumenta el número de piezas en juego, hasta llegar a 20 por jugador, dos hombres y cuatro caballeros más por cabeza.

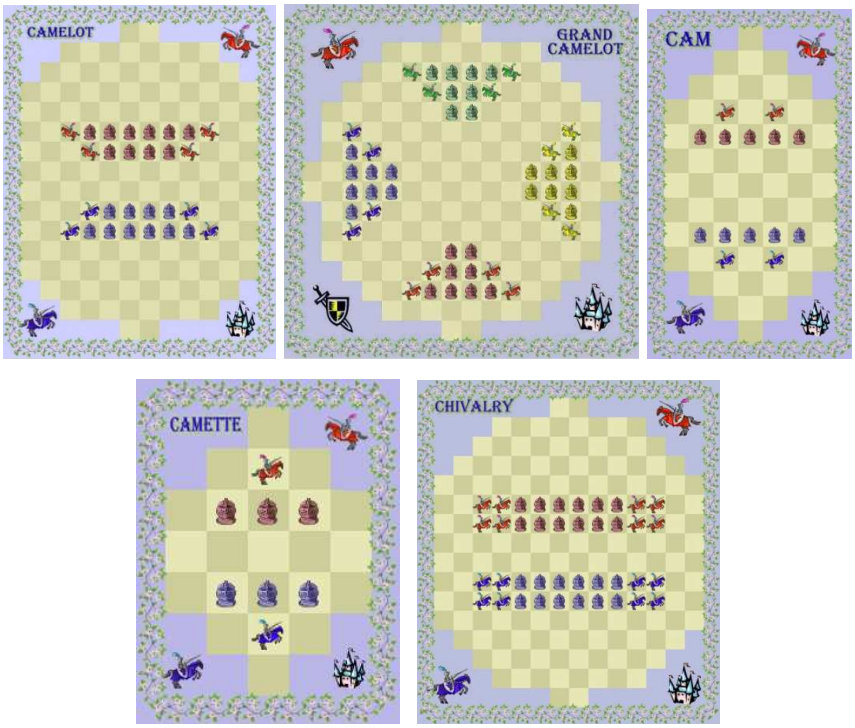


Figura 2. 16

2.5.2. CHAXX, campeón mundial de Camelot

Como se ha mencionado en el capítulo anterior, el actual programa de ordenador campeón del juego de Camelot es CHAXX, ganador del **Camelot Computer World Championship en 2008**.

Su creador, **Rick Fadden,** actualiza la versión de su programa introduciéndole pautas para mejorar su juego. La última versión, **Chaxx 1.19,** aunque todavía necesita retoques en las técnicas en el final, ya es lo suficientemente fuerte como para superar a la mayoría de jugadores humanos en la apertura y el medio juego. El trabajo actual del programa incluye una interfaz gráfica de usuario, un libro de aperturas, una determinación precisa de las relaciones Caballero-Hombre, y estrategias de finales.

2.5.3. Otros programas con mal resultado

Se han llevado a cabo una serie de intentos de programa para jugar al juego de Camelot, pero de los que no se han obtenido resultados positivos; Estos programas han participado en los campeonatos organizados por la federación anualmente y han sido incapaces de vencer a CHAXX en su última versión. (Figura 2.17):

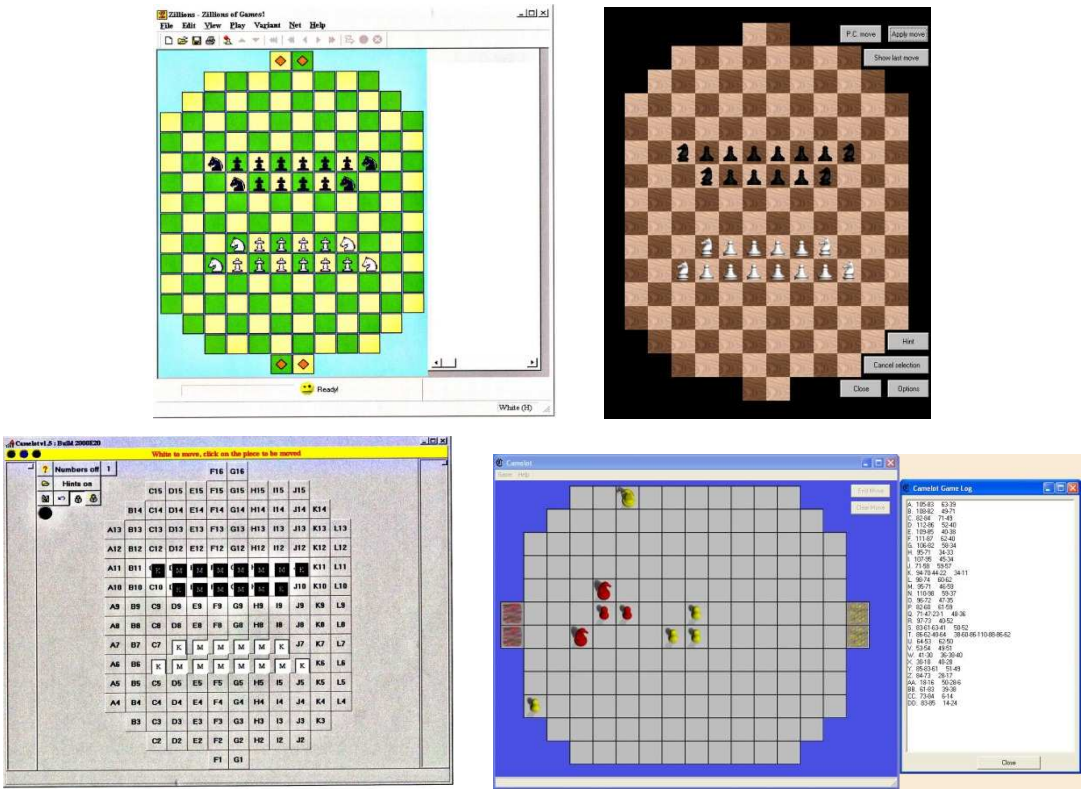


Figura 2. 17

Arriba a la izquierda se muestra el programa JENS, a su derecha se muestra la interfaz del programa llamado Daniel Garber; abajo a la izquierda se puede observar a Tima, y por último, abajo a la derecha se encuentra la interfaz en juego de Phil Knox.

2.6. Desarrollo y creación de un programa jugador de Camelot

Se han establecido en la página Web de la federación mundial de Camelot una serie de reglas, que un buen programa debe cumplir para ser competitivo y poder participar en un campeonato mundial. Debe contener tres elementos primarios y básicos:

- Una eficaz interfaz gráfica para el juego que incluya el tablero, las piezas, un contador de tiempo, los medios para guardar, pausar, etc.
- Fidelidad a las normas oficiales de juego
- Un algoritmo para realizar una evaluación precisa de la situación de la partida y de sus posiciones (función de evaluación).

A continuación se muestra un ejemplo de una representación del primer punto a cubrir (interfaz gráfica) (Figura 2.18)

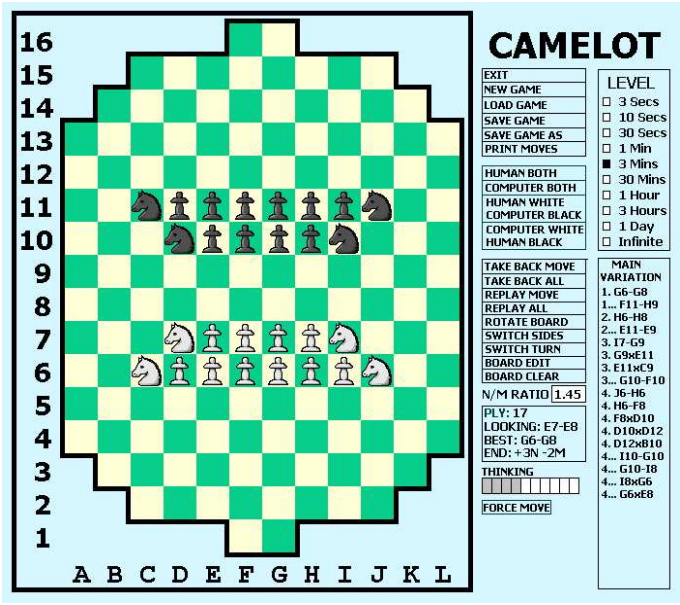


Figura 2. 18

2.6.1. Reglas para una buena evaluación de la partida

Estos son los consejos publicados en la página web oficial de la federación mundial de Camelot. Los factores que se han tenido en cuenta para el buen

funcionamiento de la función de Evaluación de este proyecto, se han obtenido a través de la práctica del juego de Camelot, y posteriormente se ha comprobado que coinciden.

1. Valores relativos de las piezas de hombre y caballero
2. Proporcionar valor a meter dos piezas en el Castillo oponente (gana el juego), que es la tercera prioridad detrás de eliminar una pieza de su propio Castillo (la más alta prioridad) y hacer una captura por medio de un salto o de una carga del caballero cuando una pieza enemiga está expuesta a la captura (la segunda más alta prioridad).
3. Valorar capturar todas las piezas oponentes manteniendo al menos dos de las tuyas (gana el juego)
4. El valor de aproximarse al Castillo enemigo
5. Valorar el número de casillas controladas por cada pieza.
6. Valorar cuando las piezas más valiosas se encuentran al frente o por detrás del resto.
7. Proximidad al centro del tablero.
8. Proximidad de los hombres a los caballeros y de los caballeros a los caballeros
9. Proximidad de las piezas hombre a las piezas hombre.
10. Exposure to a position where a piece is next to an exposed enemy piece but not exposed, itself, to the enemy piece (in other words, moving into a position where a jump on the next move will be mandatory.
11. Valorar sacar tus piezas de tu propio castillo.
12. Valorar hacer una captura a través de un salto o de una carga cuando una pieza enemiga está expuesta a ser capturada.

2.7. Conclusiones

Este capítulo recoge toda la información concerniente al juego de Camelot y a su entorno. Se trata de un juego no muy complicado y muy dinámico, fácil de jugar y de rápida conclusión, tiene muchas características que lo hacen apetecible para su juego.

La implementación de las reglas se ajustará a las establecidas por el juego oficial, y BOU deberá respetar todas las especificaciones de los movimientos de captura, aunque el usuario no se diera cuenta de si comete faltas o no.

Su programación exige una interfaz gráfica para que el usuario pueda interactuar con el código, lo hará más sencillo. Y en esta interfaz, aparte de permitir jugar un usuario contra otro, deberá incluir un apartado para jugar contra el ordenador.
Esta opción exige la inclusión de inteligencia artificial, y un procedimiento de búsqueda recursiva, que ayude en la toma de decisiones de movimientos en la partida.

Para poder evaluar el juego escoger que movimiento es mejor por parte de la maquina es necesaria una función de evaluación que estudie del estado de la partida dependiendo de una serie de factores observables sobre el tablero.

Esta interfaz además debería permitir al usuario que configure la partida a su gusto, dejando que escoja el tablero, el color de piezas, el tiempo de partida, cuando deshacer un movimiento, cuando rehacerlo y guardar estos datos siempre que desee, así como cargar partidas ya empezadas en otro momento. Siempre proporcionando un juego cómodo para el usuario.

Ya existen versiones programadas del juego de Camelot, pero la intención en este proyecto, es la de superar en juego a los ya creados mediante nuevas técnicas y análisis de factores más elaborados.

En el capítulo a continuación se detallan los objetivos planteados.

Capítulo 3. Objetivos del proyecto

A continuación se presentan y desarrollan en detalle los objetivos planteados al inicio del proyecto.

1. Programar el funcionamiento del juego de mesa Camelot.

   Se trata de un juego de táctica que requiere el planteamiento de cierta estrategia para su desarrollo. Eta decisión está motivada por el hecho de que haya poca información acerca de él. Fue editado por última vez sobre 1960, muy pocos programadores se han planteado realizar una versión interactiva, sólo uno de ellos con éxito. El trabajar sobre este juego de mesa puede llegar a ser un impulso para su desarrollo.

2. El lenguaje utilizado será Java.

3. El programa permitirá partidas Jugador vs. Jugador ó Jugador vs. BOU.

   En el momento en el que el usuario genera una partida nueva, se le da la posibilidad de escoger entre jugar con otro usuario o bien elegir como contrincante al propio programa.

4. Posibilidad de limitar el tiempo de cada jugador por turno.

   A la hora de generar una partida nueva el programa dará la posibilidad de que el usuario juegue con límite de tiempo de modo que en el momento en el que éste finalice, conseguirá la victoria aquel jugador que no haya consumido su tiempo.

5. Se podrá pausar/salvar el juego actual

   Siempre que se desee y sea cual sea la situación de la partida, se podrá parar el desarrollo del juego presionando la tecla escape, y dependiendo de la opción elegida, se puede escoger pausar de manera que el tiempo se mantiene en suspenso a la espera de que el usuario vuelva a pulsar la opción para reanudarla o bien se puede escoger salir de la partida, salvar los datos y recuperar la posición de las fichas en la siguiente partida.

6. Se podrán deshacer /rehacer movimientos

En el transcurso de la partida y simplemente pulsando un botón se podrá dar marcha atrás en el desarrollo de la partida para regresar a una jugada anterior recuperando la posición de todas las piezas. Este movimiento se podrá realizar hasta el mismo inicio de la partida. Del mismo modo, se podrá realizar un movimiento de retorno a las posiciones posteriores para retomar la que tuviéramos en el momento de decidir retroceder. Para ello, se van almacenando en una pila los movimientos realizados en el juego; en caso de realizar un cambio tras haber retrocedido, la nueva información de las piezas, sustituirá a la antigua, creándose nuevas tablas.

7. Se puede abandonar una partida en cualquier punto del juego.

Siempre que se desee, y sea cual sea la situación de la partida, puede ser abandonada por cualquiera de los dos jugadores; presionando escape y escogiendo la opción correspondiente.

8. Es posible modificar la posición inicial de las piezas al comienzo de la partida.

El programa permitirá al usuario que coloque las 28 piezas de la partida en las posiciones que desee para poder comenzar cómo quiera. También tendrá la opción de colocar un número de piezas distinto de 14 por jugador, para poder generar situaciones en estado avanzado.

9. Existe la posibilidad de cargar una partida de las dadas en una lista

Se proporciona al usuario una lista de partidas con la disposición de las piezas tal y como fueron guardadas. De éste modo, puede recuperar el transcurso de una partida abandonada, siempre y cuando se hubiera guardado anteriormente.

10. La elección de cada movimiento será hecha en cada turno con su correspondiente árbol de decisión.

Solamente para el turno correspondiente al computador, (para el usuario no se genera árbol para el movimiento de piezas), se crea un desglose de posibilidades de movimiento, varía en cada turno y siempre tiene en cuenta las posibilidades de desplazamiento del oponente.

11. Se valorará la situación del juego mediante funciones de evaluación.

Cuando se genere el árbol de decisión para el turno que corresponda a la computadora, se pasará esta función equivalente al pensamiento intuitivo humano para que evalúe la situación de la partida, y ayude en la elección de un movimiento para el jugador interactivo rival del usuario.

12. Se desarrollará una interfaz gráfica que represente la situación del juego.

Para facilitar la parte visual del desarrollo del juego, a través de herramientas de Java o bien por medios externos que luego se importen al programa, se implementará un interfaz con el que se muestren por pantalla el tablero de juego, con las piezas y las posiciones actualizadas para cada turno, como los movimientos de cada una de ellas y el tiempo real de juego.

13. Se llevará un registro de las puntuaciones máximas.

Con el fin de que los jugadores que participen en el juego por primera vez tomen una referencia de la puntuación que hayan obtenido otros, o del mismo modo, jugadores que ya hayan participado traten de superarse a sí mismos, se almacenarán las puntuaciones de las 10 mejores partidas, para conseguir una nueva forma de competición, ya no solo de derrotar al oponente, si no de superar otras partidas, en tiempo o en movimientos.

Capítulo 4. Desarrollo

Este capítulo recoge las partes concernientes al análisis, diseño y programación del sistema representadas con la mayor claridad y simplicidad posibles.

4.1. Casos de uso

Para el análisis y diseño del programa se utilizará UML (*Unified Modeling Language*), modelo utilizado para representar los sistemas, que aclara y simplifica la complejidad que en ocasiones acompaña a algunos programas. Se muestra a continuación el esquema y la descripción de los casos de uso que componen el sistema.

4.1.1. Esquema de los casos de uso

En este apartado se presenta el diagrama de casos de uso, una representación del comportamiento real del sistema y de cómo el usuario interactúa con él. Vemos en la figura 4.1, que existen dos agentes que pueden actuar sobre el sistema y cada uno de ellos tiene una serie de acciones exclusivas de su rol.

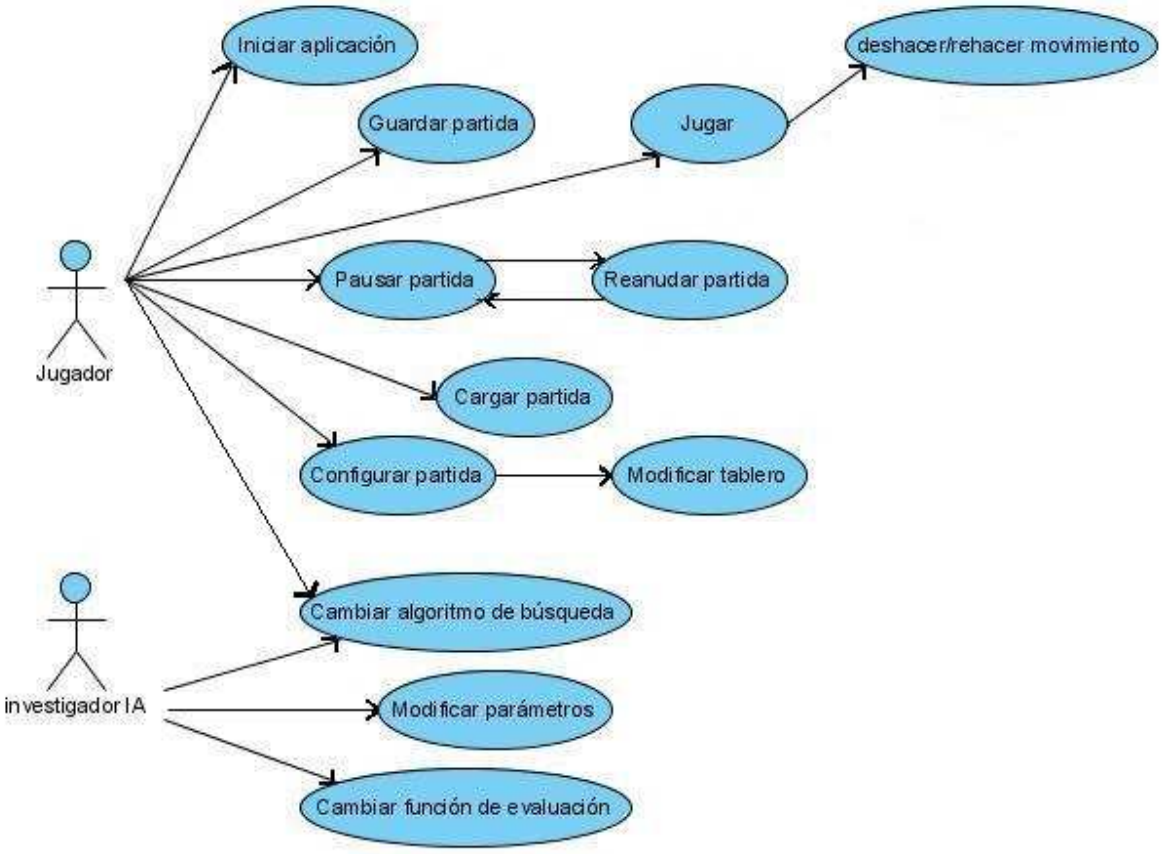


Figura 4. 1

4.1.2. Descripción textual de los casos de uso

En los siguientes apartados de este capítulo se detallaran cada uno de los casos de uso, describiendo los actores participantes, las precondiciones que se han de dar y el escenario básico en el que se encuentran.

4.1.2.1. Iniciar aplicación

Mediante este caso de uso el jugador tendrá la oportunidad de crear una cuenta de usuario, configurar el aspecto de la aplicación e introducirse en el sistema cada vez que quiera entrar a jugar, sin necesidad de que tenga que introducir sus datos posteriormente, ya que todas las modificaciones sobre los archivos de las puntuaciones u otros quedarán almacenados bajo su nombre.

Nombre	Iniciar aplicación
Actores	Jugador
Objetivo	Entrar en la aplicación
Precondiciones	-
Escenario básico	1-Introducir nombre usuario 2-Introducir contraseña 3- Pulsar entrar

Tabla 4.1

4.1.2.2. Guardar partida

O bien en el transcurso de la partida, o bien tras haber configurado un juego nuevo, mediante este caso de uso, al jugador se le presenta en la pantalla un botón, que, al ser presionado, permite que se almacenen los datos de la misma, asociada a los datos del jugador que haya entrado en el sistema. Si se presiona mientras se está desarrollando el juego, se guarda el tiempo en el que se ha pausado el juego y las posiciones que mantengan ambos jugadores (ya sean dos usuarios o un solo un jugador contra el computador), se almacenan las posiciones de todas las piezas, el color con el que juega el jugador 1 y el algoritmo de búsqueda en uso. En caso de pulsar el botón de "guardar partida" sin haber comenzado la misma, este caso de uso, simplemente guarda el nombre del jugador o jugadores que vayan a participar en el juego y la configuración que haya seleccionado para él.

Nombre	Guardar partida
Actores	Jugador
Objetivo	Guardar los datos de la partida
Precondiciones	Haber entrado en el sistema y haber configurado las preferencias del juego
Escenario básico	1.Pulsar opción de guardar partida

Tabla 4.2

4.1.2.3. Jugar

El desarrollo del juego es un conjunto de actividades que interactúan entre sí. En cuanto el jugador inicia la partida, se presenta en la pantalla el interfaz de usuario que muestra el tablero con las piezas de ambos jugadores ya colocadas y a la derecha una serie de botones que permitirán llevar a cabo diferentes acciones mientras se está jugando. En la parte superior derecha se encuentra un contador que va a marcando el tiempo que dura la partida; en caso de tratarse del modo de tiempo, este cronómetro partirá del tiempo estipulado con anterioridad hasta llegar a cero. Debajo del mismo se encuentra el botón, con el cual se puede deshacer un movimiento que se haya llevado a cabo, así como el botón que permite rehacer estos movimientos. También podemos encontrar botones para pausar la partida, retomar la partida y guardar los datos de la partida. En caso de querer abandonar el juego, es necesario guardar la partida, siempre en el turno del jugador, nunca en el turno del ordenador, para no perder los datos de juego.

Nombre	Jugar
Actores	Jugador
Objetivo	Iniciar una partida
Precondiciones	Haber entrado en el sistema y haber configurado las preferencias del juego.
Escenario básico	1.Pulsar opción de comenzar partida

Tabla 4.3

4.1.2.4. Deshacer/Rehacer movimiento

Mediante este caso de uso, tanto como si se juega contra el ordenador como si son dos jugadores, siempre durante el transcurso del juego, se pueden deshacer las jugadas que se desee hasta alcanzar la primera posición. Del mismo modo, existe otro botón en el interfaz que permite rehacer estos movimientos, hasta alcanzar la última posición llevaba a cabo. Cada movimiento realizado durante la partida se almacena en unas tablas a las que se accede en caso de querer retomar una posición.

Nombre	Deshacer/Rehacer movimiento
Actores	Jugador
Objetivo	Recuperar una posición anterior o posterior
Precondiciones	Haber entrado en el sistema y comenzar una partida
Escenario básico	1.1 Pulsar opción de deshacer 1.2 Pulsar opción de rehacer

Tabla 4.4

4.1.2.5. Pausar partida

Mediante un botón, el jugador podrá pausar el transcurso de la partida. De esta forma, el contador que controla la partida se para e inhabilita hasta nuevo aviso. Este movimiento se puede llevar a cabo en cualquier instante. En cuanto se presiona este botón, automáticamente se guarda el dato del tiempo transcurrido.

Nombre	Pausar partida
Actores	Jugador
Objetivo	Parar el tiempo del contador de partida
Precondiciones	Haber entrado en el sistema y haber comenzado una partida
Escenario básico	1.Pulsar opción de pausar partida

Tabla 4.5

4.1.2.6. Reanudar partida

El usuario, durante el transcurso del juego y siempre y cuando haya presionado anteriormente la tecla de pausar partida, podrá pulsar el botón de reanudar partida que le permite retomar el juego poniendo en marcha otra vez el contador del juego. Cuando esto ocurre, se recuperan los datos que se guardaran al presionar el botón de pausar.

Nombre	Reanudar partida
Actores	Jugador
Objetivo	El tiempo continua descontando
Precondiciones	Haber pausado la partida
Escenario básico	1.Pulsar opción de reanudar partida

Tabla 4.6

4.1.2.7. Cargar partida

El usuario, que inicialmente ha tenido que entrar en el sistema, recuperará los datos de una partida inacabada o de una partida no comenzada en la que ya se han configurado las características previamente. Según presiona el botón, al jugador se le presentará en la pantalla una lista de todas las partidas guardadas para que pueda seleccionar la que desee. Una vez escogida la partida, se cargarán los datos de la misma, y pasaremos a la pantalla de juego, con las piezas almacenadas situadas en el tablero.

Nombre	Cargar partida
Actores	Jugador
Objetivo	Cargar en pantalla una partida no finalizada
Precondiciones	Haberse entrado en el sistema
Escenario básico	1.Pulsar la opción de cargar partida 2.Escoger en la lista de almacenadas 3. Pulsar opción de cargar.

Tabla 4.7

4.1.2.8. Configurar partida

El usuario, una vez dentro del sistema, se encuentra frente una pantalla en la que se le presenta la opción de cambiar el modo del juego, establecer un tiempo límite y poder modificar el tablero.

Escoger el modo de juego consiste en elegir si la partida será entre el jugador que ha entrado contra el computador, o bien dos jugadores (uno de ellos es el jugador que ha iniciado la aplicación introduciendo su contraseña); si se escoge este modo, el usuario deberá introducir el nombre del jugador con el cual se va a enfrentar.

El usuario también puede elegir cuál será el tiempo de duración de la partida, introduciendo el número de minutos que desee. Cada jugador tiene un contador que se inicializará con el tiempo especificado. El juego se dará por finalizado en cuanto llegue a cero (para uno de los dos jugadores), y la derrota corresponderá a aquel jugador que haya consumido su tiempo.

Nombre	Configurar partida
Actores	Jugador
Objetivo	Establecer las características de una partida nueva
Precondiciones	Haber entrado en el sistema
Escenario básico	1.Pulsar opción configurar 2.Modificar el tablero (opcional) 3.Escoger el modo de juego 4.Establecer el tiempo límite de partida

Tabla 4.8

4.1.2.9. Modificar tablero

Mediante este caso de uso, el jugador, habiendo entrado previamente en el menú de configurar la partida, tiene la opción de, aparte del caso habitual en el que se comienza la partida disposición inicial oficial, pueda partir de una situación cualquiera, en la que el mismo decide el número de piezas con las que quiere comenzar el juego y las posiciones que van a tomar cada una de ellas así como las de su adversario.

Nombre	Modificar tablero
Actores	Jugador
Objetivo	Establecer la posición de las piezas
Precondiciones	Entrar en el sistema e introducirse en configurar partida
Escenario básico	1. Pulsar opción de modificar tablero 2. Pinchar sobre las piezas y soltarlas en el tablero 3. Pulsar Ok/Cancel

Tabla 4.9

4.1.2.10. Cambiar algoritmo de búsqueda

Mediante este caso de uso, el usuario, que en este momento actúa como investigador de inteligencia artificial, tiene la opción, de cambiar como desee el algoritmo de búsqueda utilizado para obtener los movimientos óptimos en el juego de la computadora en el transcurso de las partidas. El usuario con el rol de jugador, habiendo entrado con su contraseña en el sistema, también puede modificar el algoritmo de juego en la configuración de la partida. Tras esto, se almacenan los cambios que se hayan llevado a cabo en el código.

Nombre	Cambiar algoritmo de búsqueda
Actores	Investigador IA, Jugador
Objetivo	Modificar el algoritmo de búsqueda del árbol de decisión
Precondiciones	Haber entrado en el sistema como investigador IA
Escenario básico	1.Pulsar opción de cambiar algoritmo de búsqueda

Tabla 4.10

4.1.2.11. Modificar parámetros

Con este caso de uso, habiendo pulsado en un menú la opción correspondiente, el investigador de inteligencia artificial, habiendo entrado en el sistema anteriormente, puede cambiar los parámetros de los métodos del juego, modificando de este modo el código, para finalizar guardando los cambios realizados sobre él.

Nombre	Modificar parámetros
Actores	Investigador IA
Objetivo	Cambiar los parámetros del código
Precondiciones	Haber entrado en el sistema como investigador IA
Escenario básico	1.Pulsar opción de modificar parámetros

Tabla 4.11

4.1.2.12. Cambiar función de evaluación

Como investigador de inteligencia artificial, habiendo introducido previamente un nombre de usuario con una contraseña adecuada, el usuario tiene la opción, habiendo presionado el botón correspondiente, de modificar el código de la aplicación para cambiar a su gusto la función que evalúa el estado del juego en cada turno. Esta función depende de una serie de características que a su vez también se pueden modificar, al igual que los factores de los que dependen. Finalmente se guardan los cambios realizados sobre el código.

Nombre	Cambiar función de evaluación
Actores	Investigador IA
Objetivo	Modificar la función de evaluación utilizada en el código
Precondiciones	Haber entrado en el sistema como investigador IA
Escenario básico	1.Pulsar opción cambiar función de evaluación

Tabla 4.12

4.1.3. Organización de los casos de uso en paquetes

La distribución de estos casos se puede agrupar en pequeños grupos tal y como se presentan ante el usuario. De este modo podemos decir que, el primer caso de uso, "iniciar aplicación", forma el paquete 0 que aparece en primer lugar para permitir al usuario entrar al sistema. Posteriormente, los casos de uso de "Cargar partida", "cambiar algoritmo" y "configurar partida", que le permiten configurar el juego o abrir uno ya guardado, forman el paquete 1. De este mismo, se bifurca el paquete 2, con el único caso de uso de "modificar tablero". El paquete 3, está constituido por los casos de "Jugar", "Deshacer/rehacer movimiento", "Pausar partida", "reanudar partida" y "guardar partida"; y por último el paquete 4 con los casos de uso de "Cambiar algoritmo de búsqueda", "Cambiar función de evaluación" y "Modificar parámetros" para el usuario investigador**.**

A continuación (Figura 4.2) se muestra un pequeño esquema de las relaciones entre los paquetes que acabamos de describir (es una demostración de cómo se presenta la navegación de pantallas al inicio del programa)

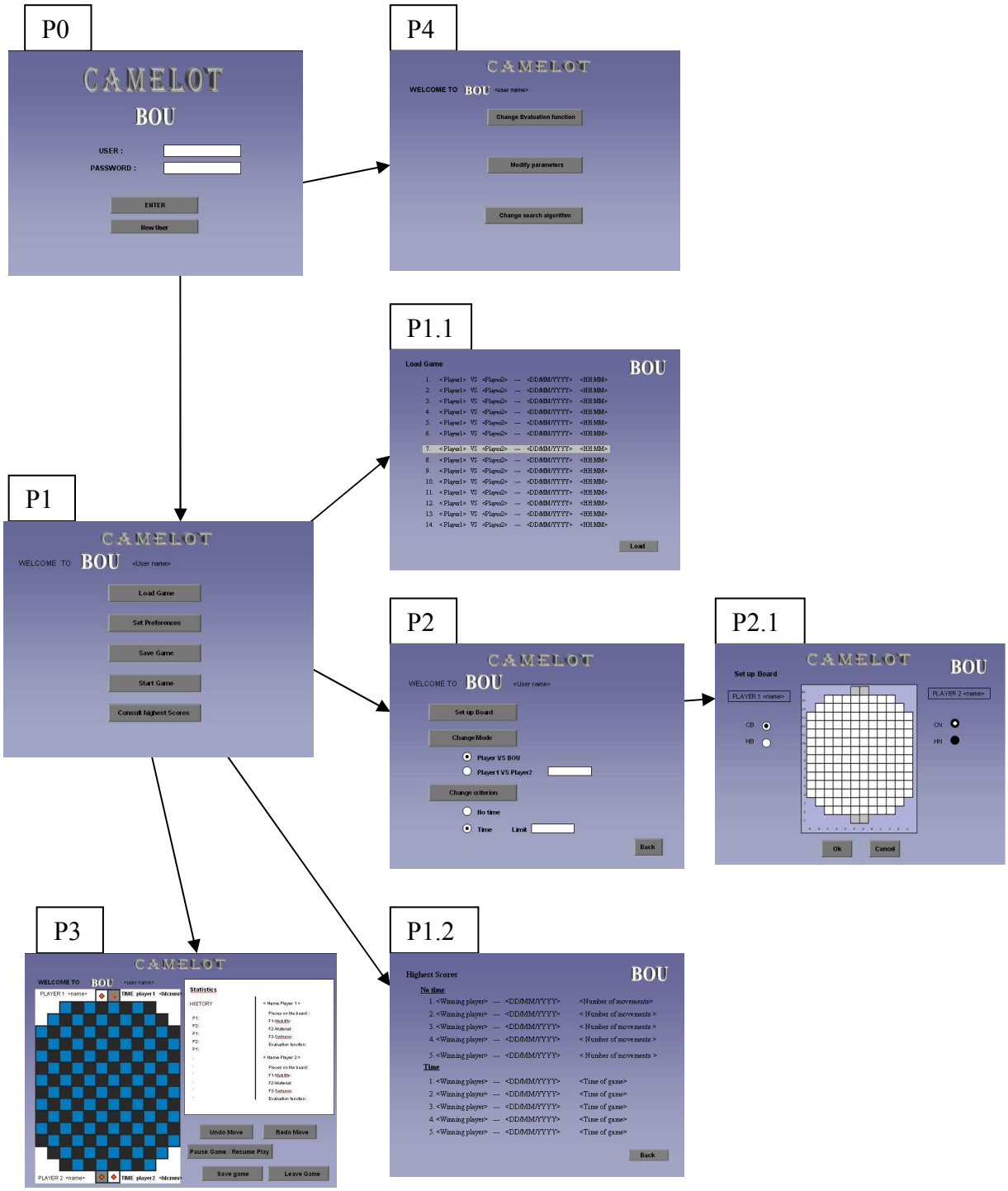

P0
CAMELOT
BOU
USER :
PASSWORD :
ENTER
New User
P4
CAMELOT
WELCOME TO BOU
Change Evaluation function
Modify parameters
Change search algorithm
P1.1
Load Game
BOU
Load
P1
CAMELOT
WELCOME TO BOU
Load Game
Set Preferences
Save Game
Start Game
Consult highest Scores
P2
CAMELOT
WELCOME TO BOU
Set up Board
Change Mode
Player VS BOU
Player1 VS Player2
Change criterion
No time
Time
Limit
Back
P2.1
CAMELOT
BOU
Set up Board
Ok
Cancel
P3
CAMELOT
WELCOME TO BOU
Statistics
HISTORY
Undo Move
Redo Move
Pause Game
Resume Play
Save game
Leave Game
P1.2
Highest Scores
BOU
No time
Time
Back

Figura 4.2

4.2. Requisitos funcionales de usuario

A continuación se procede a mostrar una breve descripción de los requisitos de usuario más importantes especificados en el capítulo de objetivos (Capítulo 3) de este documento.

ID REQUISITO	REQ-4
NOMBRE REQUISITO	**Jugar partidas Player vs. Player o Player vs. CPU**
CASOS DE USO	-Configurar partida
DESCRIPCIÓN	La aplicación permite configurar la partida de modo que se puede elegir entre enfrentar a dos jugadores o jugar contra la máquina. Esta selección se lleva a cabo a través del menú de configurar partida y configurar modo, en el que se recogen las dos opciones, marcando una casilla.

Tabla 4.13

ID REQUISITO	REQ-5
NOMBRE REQUISITO	**Limitar tiempo de partida**
CASOS DE USO	-Configurar partida
DESCRIPCIÓN	Se establece un límite en la duración de las partidas, dándolas por finalizadas en cuanto el contador de uno de los jugadores llegue a 0. Para ello, a través del caso de uso de configurar partida, en la opción de tiempo, se introduce el tiempo deseado de duración. (También es posible escoger una opción de partida sin tiempo límite)

Tabla 4.14

ID REQUISITO	REQ-6
NOMBRE REQUISITO	**Pausar la partida**
CASOS DE USO	-Pausar partida
DESCRIPCIÓN	En cualquier momento en el trascurso de la partida, un jugador puede pausar el tiempo, pero sólo el asociado a su contador; Su contrincante podrá pausar el suyo del mismo modo. Posteriormente se habilitará así la opción de reanudar partida.

Tabla 4.15

ID REQUISITO	REQ-7
NOMBRE REQUISITO	**Reanudar la partida**
CASOS DE USO	-Reanudar partida
DESCRIPCIÓN	En cualquier momento en el trascurso de la partida, un jugador puede reanudar el juego, siempre y cuando se haya pulsado previamente la opción de pausar partida. Ésta, se retomará tal y como quedó al pausarla.

Tabla 4.16

ID REQUISITO	REQ-8
NOMBRE REQUISITO	**Guardar la partida**
CASOS DE USO	-Guardar partida
DESCRIPCIÓN	La aplicación permite almacenar los datos de una partida que se haya configurado previamente. Puede llevarse a cabo durante el transcurso de la misma. Se realiza mediante el caso de uso de guardar partida y almacena el nombre del jugador que entró en la aplicación y el del jugador contra el que juega, su color de piezas, el tiempo de partida y la disposición de las figuras sobre el tablero

Tabla 4.17

ID REQUISITO	REQ-9
NOMBRE REQUISITO	**Deshacer/ Rehacer movimientos**
CASOS DE USO	-Deshacer/rehacer movimiento
DESCRIPCIÓN	Este requisito queda cubierto mediante los casos de uso de deshacer y rehacer movimiento, mediante los cuales, se pueden recuperar todas las posiciones pasadas que se han realizado desde el inicio de la partida, así como volver a rehacerlas

Tabla 4.18

ID REQUISITO	REQ-10
NOMBRE REQUISITO	**Abandonar partida**
CASOS DE USO	-
DESCRIPCIÓN	El programa deberá permitir abandonar la partida en cualquier momento de su desarrollo. No existe ningún caso de uso asociado, porque esa actividad se lleva a cabo mediante la sencilla pulsación de un botón.

Tabla 4.19

ID REQUISITO	REQ-11
NOMBRE REQUISITO	**Modificar posición inicial de las piezas**
CASOS DE USO	-Modificar tablero
DESCRIPCIÓN	Mediante el caso de uso de modificar tablero, el jugador puede configurar la posición que tendrán todas las piezas al comienzo de la partida. Para ello tiene que escoger la opción de *modificar tablero* y se le presentará un tablero vacío en el que podrá situar las piezas en las casillas que guste.

Tabla 4.20

ID REQUISITO	REQ-12
NOMBRE REQUISITO	**Realizar árbol de decisión**
CASOS DE USO	-Cambiar algoritmo de búsqueda
DESCRIPCIÓN	Se escoge el algoritmo de búsqueda y en cada turno de la maquina se crea un árbol de decisión, para comprobar las diferentes opciones que posee.

Tabla 4.21

ID REQUISITO	REQ-13
NOMBRE REQUISITO	**Evaluar con función de Evaluación**
CASOS DE USO	-Cambiar función de evaluación
DESCRIPCIÓN	El usuario escoge el algoritmo que desea para que éste devuelva la mejor opción de todo el abanico de posibilidades desplegado por el árbol de movimientos, descartando aquellos nodos con función de evaluación menor en caso de nivel que maximice y mayor en caso de nivel que minimice.

Tabla 4.22

ID REQUISITO	REQ-14
NOMBRE REQUISITO	**Registrar las puntuaciones máximas**
CASOS DE USO	-
DESCRIPCIÓN	Se almacenan los datos de aquellas partidas jugadas, los relacionados con el tiempo transcurrido y con la puntuación obtenida. Este requisito se cubre automáticamente, al finalizar cada juego. Estas puntuaciones se guardan en dos archivos diferentes, y ambos son visibles por el usuario antes de configurar una partida nueva.

Tabla 4.23

4.3. Las Clases de BOU

El diagrama de clases es otro de los muchos modelos que forman parte de UML y que nos permiten observar las clases del sistema con sus relaciones estructurales y herencia. A continuación, en los siguientes apartados se muestra al usuario, un esquema gráfico de las clases de este sistema así como una breve descripción de aquellas más importantes y de sus métodos más significativos.

4.3.1. Esquema de clases

La representación de este sistema es la siguiente:

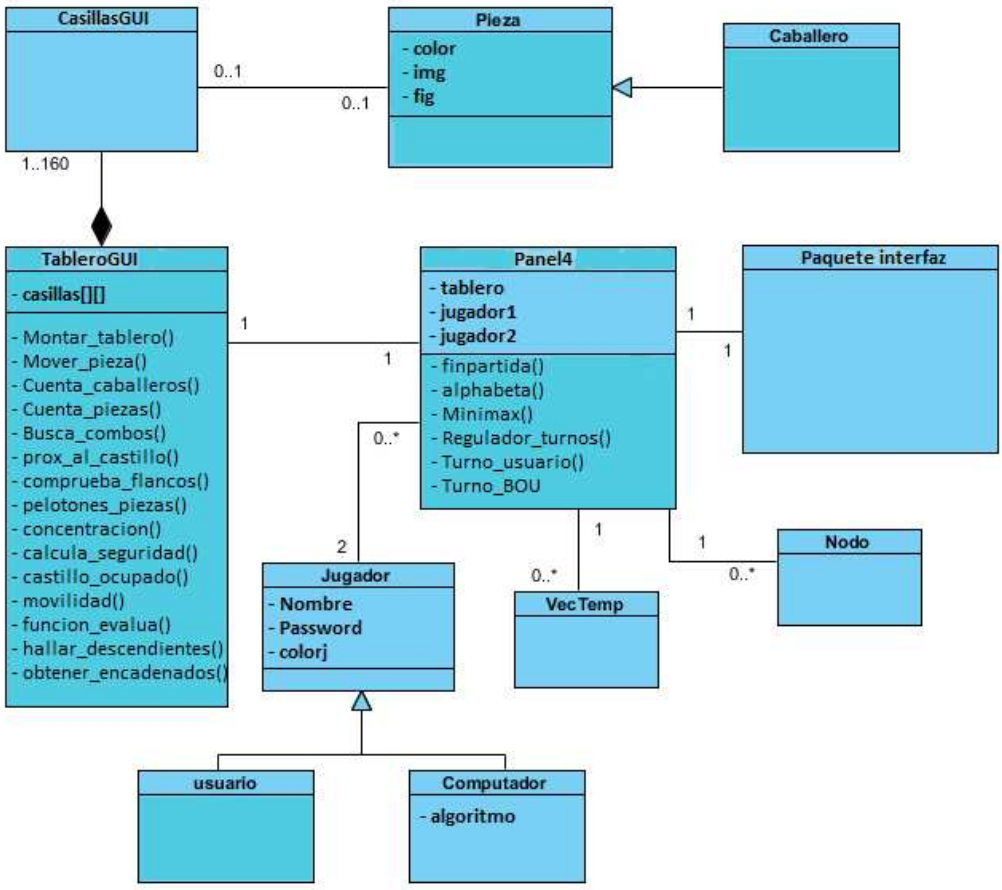


Figura 4. 3

En la figura 4.3 podemos observar una de las clases con el nombre de "Paquete interfaz". Este paquete, consta de 7 clases que no participan en las relaciones de las más significativas, por tanto se han excluido del diagrama. Cabe a destacar entre las clases de la interfaz gráfica a Panel4, sobre la que recae todo el peso del funcionamiento de la partida, y posee relaciones con el resto de clases del diagrama.

4.3.2. Descripción textual de las clases

Las clases de mayor importancia, son *Panel4* (perteneciente paquete de clases de la interfaz) y *TableroGUI*. En segundo lugar, cabe destacar *VecTemp*, que es la clase del vector que almacena cada movimiento. A continuación se detalla una lista de las funciones que las componen, así como una breve explicación de las principales.

4.3.2.1. La clase TableroGUI

Esta clase es la encargada del buen funcionamiento de la capacidad de decisión de BOU programa. Esta clase se forma por un jFrame que contiene una matriz de dimensiones 16 y 12 de elementos casillasGUI, que son pequeños jPanel que poseen la función de identificar cada uno de ellos, y contiene un fondo y la capacidad de dar cabida a una pieza. Posee tanto funciones tan importantes como la evaluación del juego, como comprobación de normas, rangos, reglas básicas que le dan oficialidad al juego. A continuación se presenta una breve descripción de los métodos más significativos que la componen

- **Tablero():** constructor de la clase Tablero, crea un objeto tablero, y monta las casillas que lo componen, estableciendo en cada posición el fondo que corresponde, como indica en la figura 4.4.

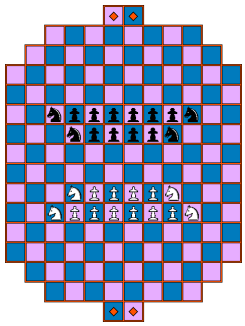


Figura 4. 4

- **Mover_multiple**: esta función se encarga de identificar cuando un movimientos es encadenado (observando el contenido de la variable *enc* y la variable *sig* para ver el movimiento siguiente)y lo lleva a cabo hasta que la variable *sig* es igual a null.
- **Mover_pieza**: Mueve una pieza en el tablero, siempre u cuando no sea un movimiento encadenado (la variable *enc* posee el valor FALSE)

- **Busca_salir_castillo**: esta función, en caso de encontrarse una pieza en su propio castillo, evalúa las posiciones libres contiguas, para abandonar la posición y escoge la mejor de ellas
- **Vaciar_su_castillo**: esta función, en caso de que sea BOU el que juega, y se produzca una ocupación temporal del castillo amigo por una de sus piezas, descarta desglosar el árbol de búsqueda y pone en movimiento esa pieza, buscando el movimiento que le resulte más beneficioso.
- **Cuenta_caballeros**: es uno de los factores de la función de evaluación; cuenta el número de caballeros en juego. Para verlo más en detalle, consultar el apartado 4.5.1.3. de este capítulo
- **Cuenta_piezas**: es uno de los factores de la función de evaluación; cuenta el número de piezas en juego. Para verlo más en detalle, consultar el apartado 4.5.1.1. de este capítulo.
- **Evalua_factor**: esta función es la encargada de llamar a todos y cada uno de los factores que componen la función de evaluación y devuelve su valor final
- **Comprobar_rango**: esta función se encarga de evaluar que el movimiento introducido por teclado por el usuario, cumple con todas las condiciones de juego pertinentes (que las casillas origen y destino existen y que el origen contiene pieza del color que corresponde y la casilla destino no).
- **Busca_combos:** es uno de los factores de la función de evaluación; busca combos de piezas caballero y hombre en casillas contiguas. Para verlo más en detalle, consultar el apartado 4.5.1.10. de este capítulo.
- **Prox_al_castillo**: es uno de los factores de la función de evaluación; evalúa la proximidad de las `piezas al castillo enemigo, y la concentración de ellas Para verlo más en detalle, consultar el apartado 4.5.1.4. de este capítulo.
- **Comprueba_flancos**: es uno de los factores de la función de evaluación; analiza si existen piezas enemigas, aproximándose por los flancos al castillo. Para verlo más en detalle, consultar el apartado 4.5.1.5. de este capítulo.
- **Pelotones_piezas**: es uno de los factores de la función de evaluación; evalúa si hay piezas amigas que avanzan al castillo de manera conjunta. Para verlo más en detalle, consultar el apartado 4.5.1.6. de este capítulo.

- **Concentración**: es uno de los factores de la función de evaluación; se encarga de comprobar cuán juntas están las piezas para evaluar su capacidad de defensa. Para verlo más en detalle, consultar el apartado 4.5.1.8. de este capítulo.
- **Calcula_seguridad**: es uno de los factores de la función de evaluación. Se encarga de comprobar cuantas piezas amigas y enemigas hay en territorio amigo y calcula la relación entre ambas. Para verlo más en detalle, consultar el apartado 4.5.1.2. de este capítulo.
- **Castillo_ocupado**: es uno de los factores de la función de evaluación. Simplemente comprueba si el castillo enemigo ha sido tomado por una de tus piezas. Para verlo más en detalle, consultar el apartado 4.5.1.7. de este capítulo.
- **Cuenta_movilidad**: Esta función comprueba de manera individual el número de movimientos que una pieza dentro de un tablero puede realizar. Devuelve un número entero.
- Movilidad: es uno de los factores de la función de evaluación; calcula el número de movimientos totales que pueden realizar todas las piezas de un mismo color en todo el tablero. Para verlo más en detalle, consultar el apartado 4.5.1.9. de este capítulo.
- **Encycapt**: este método controla que un movimiento encadenado termina en captura, ya que, un movimiento encadenado puede estar formado por una sucesión de saltos sin captura. De este modo, localizamos aquellos encadenados, haciendo un seguimiento de todos sus pasos, que terminan capturando alguna pieza enemiga, para seleccionarlos como prioritarios en el árbol de búsqueda.
- **Escoger_capturas**: este método que resulta ser bastante importante, forma parte de la función de Minimax, Alphabeta y F-Alfhabeta, e identifica en la lista de todos los movimientos que las piezas de un color pueden realizar, aquellos que son capturas y los guarda en un listado aparte. Se indica en el apartado 4.7.1.1. de este capítulo la importancia de esta función
- **Hay_capturas**: Recorre la lista total de los movimientos que las piezas de un color pueden realizar en el tablero e identifica, observando la variable *capt*

en cada una de ellas, si existen capturas entre ellas, devolviendo de esta manera un TRUE en caso de ser positivo, o FALSE en caso de no encontrar ninguno

- **Función_evalua:** es la función de evaluación del programa BOU, utilizada en los algoritmos de búsqueda del programa. Llama a los nueve factores que la componen y devuelve el valor resultante de combinarlos con sus pesos correspondientes.
- **ya_enc**: Esta función, en el proceso de búsqueda de movimiento, va comprobando para cada nuevo, si se encuentra ya en la lista de los ya guardados, para evitar tenerlos por duplicado
- **busca_movs**: Dada una pieza en una casilla, comprueba las 8 posibles movimientos que podría realizar u aquellos que puede llevar a cabo, los devuelve listados
- **obtener_encadenados**: Este método, partiendo de los 8 movimientos simples que una pieza puede realizar, identifica cuáles de ellos son capturas o saltos y analiza los siguientes movimientos que se podrían realizar de manera encadenada. Devuelve un listado de listas con todos ellos.
- **hallar_descendientes**: Esta función, que forma parte de los algoritmos de búsqueda, es la que se encarga de hallar los descendientes de cada una de las piezas del mismo color que se encuentran en el tablero en ese momento. Toma todas las piezas del mismo color, encuentra sus posibles movimientos simples, y a su vez, si es posible, los encadenados, componiéndolos en una lista de listas, para posteriormente devolverlos. Tiene un peso bastante importante ya que utiliza, para obtener un listado de todos movimientos posibles (incluyendo los encadenados), de otros métodos tales como *busca_movs*, *obtener_encadenados*, *montar_encadenados*, *inicializar_pila*, etc. A continuación se muestra un pequeño esquema del funcionamiento de esta importante función (figura 4.5):

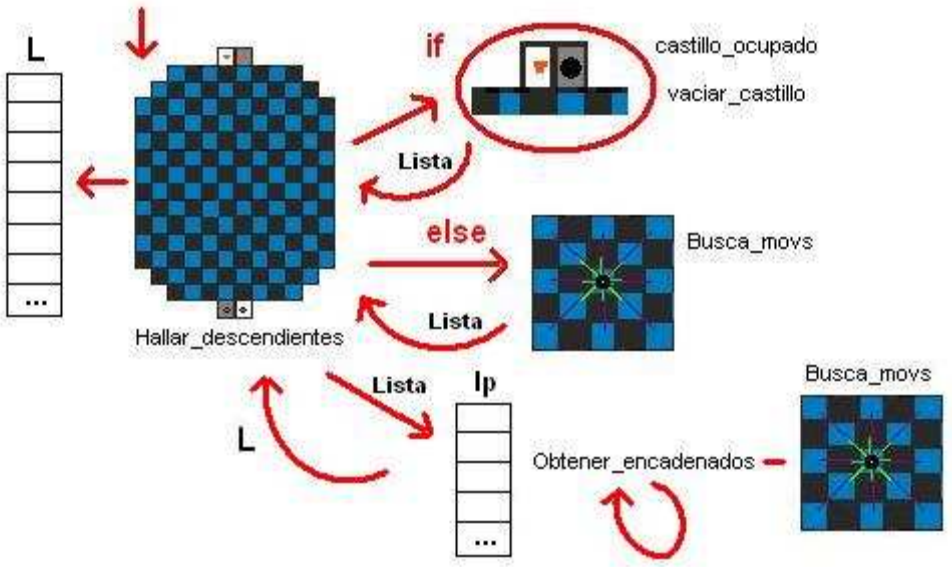


Figura 4. 5: comprueba si el castillo propio está ocupado, si lo está, se obliga a desocupar esa casilla; de no ser el caso, con *busca_movs*, obtenemos una lista con todos los movimientos simples y posteriormente con *obtener_encadenados*, se devuelve una lista L con todas las posibilidades de movimientos, incluidos los encadenados, limpia de duplicados y movimientos prohibidos .

4.3.2.2. La clase Panel4

Es la clase principal del paquete de la interfaz del juego. Se explican en más detalle en el capítulo dedicado a la interfaz del juego (). Como se puede observar en el diagrama de clases de la figura anterior, el paquete interfaz soporta el peso de numerosas clases que confluyen para iniciar la partida, ya que ésta, es el motor que mueve el programa, contiene la función de *iniciar_partida,* que arranca el juego y posee métodos tan importantes como aquellas que se encargan del almacenamiento de los movimientos, los algoritmos de búsqueda para la creación del árbol de decisión, las condiciones de fin de partida, la configuración inicial de juego, la pausa y reanudación del juego y el reseteo de los contadores que lo controlan, así como la capacidad de deshacer y hacer movimientos, todo aquello que componen los casos de uso del usuario. A continuación se muestra un listado de sus atributos, todos los métodos que la componen así como una breve descripción de aquellos más significativos.

- **Fin_partida**: esta función es vital para el desarrollo de la partida, ya que evalúa que no se hayan dado las condiciones de fin de partida. Comprueba que hay más de dos piezas de cada color y que ninguno de los dos castillos este ocupado por piezas enemigas. Se utiliza al final de cada jugada y de cada ronda, para que finalice el juego en cuanto se identifiquen estas condiciones.

- **Evalua_movimiento:** esta función recorre el tablero evaluando las diferentes posibilidades de todos los movimientos de BOU. Se encarga de escoger el mejor de todos los movimientos proporcionados por recorre_arbol; es la función que devuelve el movimiento finalista; escoge aquel movimiento que proporcione más valía para su estatus. Este método actúa como un algoritmo de búsqueda a profundidad 1, no tiene en cuenta la contrarrespuesta del contrario, simplemente evalúa en el momento actual de la partida, qué jugada, le beneficiaría más.
- **Regulador_turnos:** esta está encargada de coordinar los turnos en la partida, controla a través de un entero que le indica el tipo de partida en la que nos encontramos, y va cambiando del jugador 1 al jugador 2, llamando al método que corresponda, "Turno_BOU" ó "Turno_usuario". También hace las modificaciones pertinentes en la interfaz de juego en cada movimiento realizado.
- **Turno_BOU:** esta función es la encargada de llevar a cabo los movimientos de BOU; teniendo en cuenta el algoritmo que se haya escogido en la configuración de la partida, genera un movimiento, lo lleva a cabo y lo guarda en el registro de desplazamientos.
- **Turno_usuario**: este método se encarga del movimiento de usuario; muestra un mensaje por pantalla a través de un elemento jOptionPane, que avisa al jugador que introduzca un movimiento y permaneces a espera de que se recoja a través del teclado, para proceder después a validarlo y mover la pieza escogida.
- **Falphabeta**: es uno de los algoritmos de búsqueda utilizados en el proceso de evaluación de BOU. Para más información, consultar la sección 2.2.2.4.
- **Alphabeta**: es uno de los algoritmos de búsqueda utilizados en el proceso de evaluación de BOU. Para más información, consultar la sección 2.2.2.3.
- **Minimax:** es uno de los algoritmos de búsqueda utilizados en el proceso de evaluación de BOU. Para más información, consultar la sección 2.2.2.1.

4.3.2.3. La clase VecTemp

Esta clase se encarga de almacenar cada uno de los movimientos y todos los atributos asociados a ellos, tales como son, si se trata de un movimiento encadenado, o una captura, o un salto…atributos que los distinguen entre sí, y son decisivos a la hora de escoger el mejor de ellos en el árbol de búsqueda. Cabe a destacar dos de ellos:

- Score: Es el valor principal de esta clase se debe almacenar en una variable float. Se trata del valor de la función de evaluación del tablero obtenido al realizar el movimiento, que también se indica. Se ha escogido una variable float ya que, cada uno de los factores que componen la función de evaluación pueden contener decimales. Esta variable es decisiva ya que es el punto de comparación en el árbol de búsqueda.
- Num: esta variable contiene un entero que se inicializa a 0. Solamente cambia su valor cuando se guarda el movimiento llevado a cabo, tanto por el usuario como por BOU. Gracias a esta variable se identifica el carácter de la figura que, en caso de un movimiento de salto, haya sido capturada. Puede tomar los siguientes valores:

  -1: si la pieza capturada ha sido un hombre blanco.

  -2: si la pieza capturada ha sido un caballero blanco.

  -3: si la pieza capturada ha sido un hombre negro.

  -4: si la pieza capturada ha sido un caballero negro.

  -0: permanece así, mientras el movimiento sea simple, o un movimiento de *galope* sobre una pieza de su mismo color.

4.4. Observaciones sobre el tablero

Antes de comenzar a jugar, , cabe a destacar una serie de características sobre el tablero, y sobre situaciones producidas en el desarrollo de las partidas, que nos ayudarán a encontrar las claves para conseguir la función de evaluación adecuada para conseguir la victoria.

4.4.1. Representación canónica de las piezas

La posibilidad de realizar la *Carga del caballero*, les proporciona a los caballeros un valor adicional respecto al de los hombres, ya que estas piezas, te

permiten avanzar más deprisa por el tablero para alcanzar el castillo y a la vez pueden provocar la ruptura de una defensa enemiga.

De este modo, si representamos una pieza hombre con la letra **H,** y quedando el caballero representado con la letra **C,** la relación de ambas piezas sería la siguiente:

$$H= k.C$$

Donde la constante k toma el valor de **0.75** (H=0,75C)
A continuación, explicamos la procedencia del valor asignado a k.

4.4.1.1. El valor superior del caballero, k

Tomando tres situaciones hipotéticas cualesquiera se analiza el comportamiento en el tablero de ambas piezas para poder comparar su valor en cada una de las partidas. Se presentan en las figuras a continuación una serie de piezas distribuidas por el tablero, y los movimientos hipotéticos que llevarían a cabo la pieza de hombre y la pieza de caballero. Aunque la disposición de las piezas que participan en los distintos movimientos encadenados no sea exactamente la misma para ambos, se provoca el mismo resultado en cuanto a proximidad al castillo, para que ambos casos estén en igualdad de condiciones.

Situacion1: Casilla extremo inferior situada en la distancia más alejada del castillo oponente

Puede corresponder con el extremo izquierdo o derecho del tablero y se comprueba que:

-El hombre: tomando como referencia el movimiento que le permite avanzar más rápidamente y descartando capturas de jugadores enemigos, esta pieza tardaría 7 movimientos en alcanzar el objetivo del castillo, como se observa en la figura 4.6.

-El caballero: en el avance hacia el castillo oponente, y descartando cualquier intrusión de las piezas enemigas, el caballero puede alcanzar el castillo objetivo en 7 movimientos, de la misma manera que el hombre, pero, contando con su movimiento exclusivo de la *carga del caballero*, este tiene permitido avanzar sobre cuantas piezas amigas desee y después capturar tantas piezas enemigas como le sea posible. Por tanto,

como observamos en este ejemplo, el caballero, suponiendo la situación más favorable que se nos puede plantear, tarda 7 movimientos y además, ha eliminado a 6 piezas enemigas del tablero, lo que supone un valor muy superior del tablero resultante

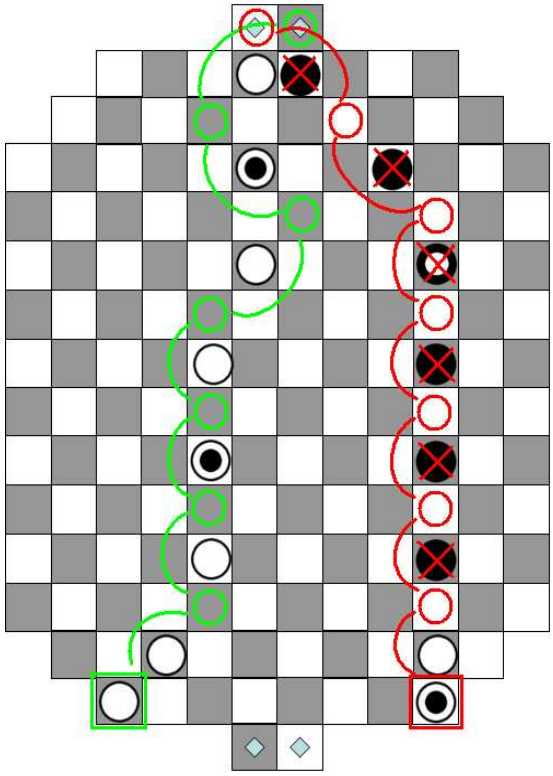


Figura 4. 6

Situación 2: Casilla extremo lateral de la parte superior del tablero

Puede tratarse del lateral izquierdo y el derecho, el comportamiento es el mismo.

-Hombre: Como en la situación anterior, tomando la situación del tablero como la más favorable que se puede presentar, siempre buscando la distancia más rápida, la pieza llega al castillo en 2 movimientos realizando el movimiento de *galope* sobre dos piezas de su mismo color.

-El caballero: a éste, le toma el mismo número de movimientos que a una pieza de hombre llegar a su objetivo, 2. Pero, en la situación más optimista en la que, alguna pieza de su mismo equipo y un oponente ocupan las casillas de su recorrido, puede llevar a cabo la *carga del caballero*, alcanzando el castillo oponente en 2 movimientos y captura la pieza enemiga, como se observa en la figura 4.7

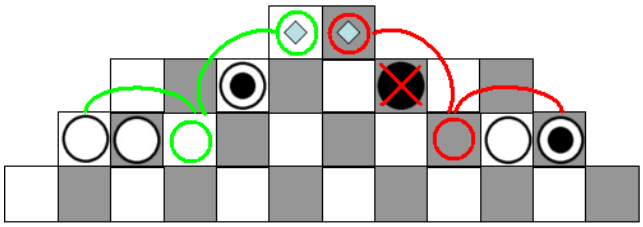


Figura 4. 7

Situación 3: Posición central del tablero, en la parte más alejada al castillo oponente:

-Hombre: la distancia al objetivo por el camino más corto, tanto si te encuentras en la casilla izquierda como en la derecha, siempre partiendo de la situación más favorable que se puede producir, es de 3 movimientos, sin capturar ninguna pieza enemiga y realizando *galope* sobre las piezas de su color.

-Caballero: en la situación más optimista, gracias a la carga del caballero, puede alcanzar el objetivo en 3 movimientos, como la pieza de hombre, pero, a su vez, capturando a dos piezas enemigas, gracias al movimiento de la *carga*. Podría encadenar dos movimientos de *galope* sobre piezas amigas y capturar solo un oponente, o como el hombre, saltar sobre tres piezas de su mismo color, pero, como hemos dicho, planteamos la situación que más favorecería al caballero (Figura 4.8).

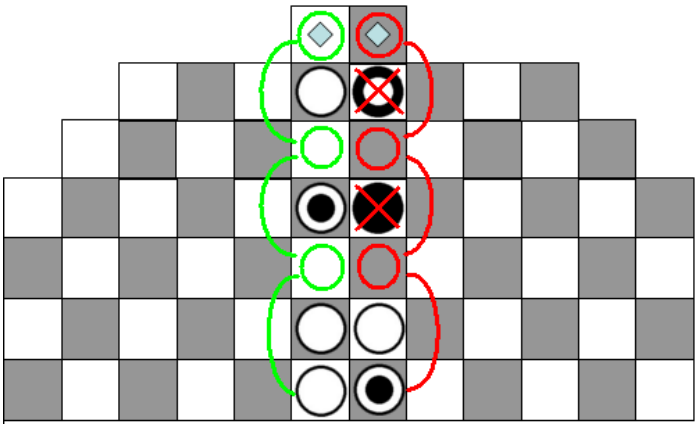


Figura 4. 8

Para justificar el valor superior del caballero, nos basamos ahora en dos de los factores que componen la función de evaluación del tablero; dos de los que poseen más peso de todos ellos, **número de piezas** (para más información consultar el capítulo 4.5.1.1) y **proximidad al castillo** (para más información consultar el capítulo 4.5.1.4).

Ambos, a pesar de tener mucha importancia poseen diferente peso en la función; la proximidad al castillo cuenta con un valor cuantitativo de 200, y el número de piezas, vital para la resolución de la partida, con un valor de 200 también. (Para más información acerca de los pesos de los factores de la función de evaluación, consultar el capítulo 4.5.2.).

Es decir, a continuación se procede a evaluar el valor de estos movimientos sobre el tablero, teniendo en cuenta todas las piezas del tablero, una vez que se ha llevado a cabo el movimiento (cada uno por separado) y descartando el resto de factores que componen la función de evaluación. Es decir, aplicamos esta fórmula:

Func.ev= 200*(f.proximidad al castillo)+ 200(f.numero de piezas)

Situacion1: Casilla extremo inferior situada en la distancia más alejada del castillo oponente

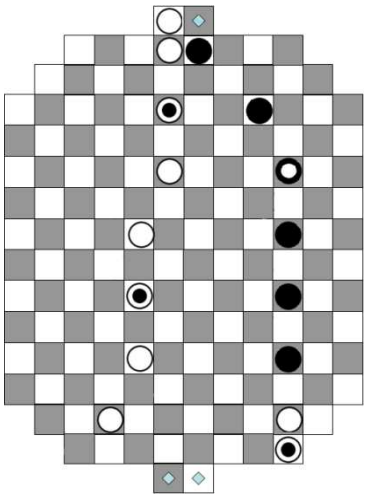


Figura 4. 9

f.prox_cas $_{blancas}$ =21000-((76*100) + (1400*4)) =7800

f.prox_cas $_{negras}$ =21000-((54*100) + (1400*8))=4400

Según el cálculo de los factores de la función de evaluación se aplica la fórmula para obtener el valor de este factor:

f.prox=(f.prox_cas $_{blancas}$- f.prox_cas $_{negras}$) / (f.prox_cas $_{blancas}$+ f.prox_cas $_{negras}$+1)

f.prox= 0,278

f.num_piez $_{blancas}$=10

f.num_piez $_{negras}$= 6

Aplicamos la fórmula anterior:

$f.num\_piez=(f.num\_piez_{blancas}- f.num\_piz_{negras}) / (f.num\_piez_{blancas}+ f.num\_piez_{negras}+1)$

f.num_piez= 0,235

Por tanto, **func.ev**=200*0.278 + 200*0.235= **102,6**

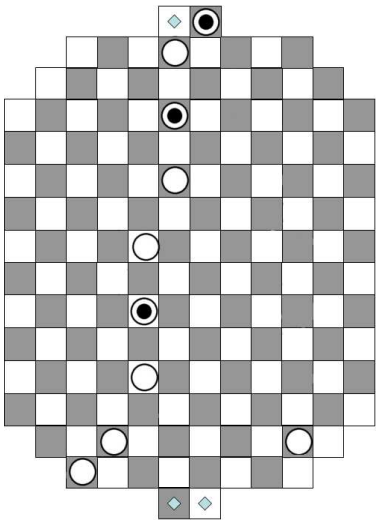


Figura 4. 10

$f.prox\_cas_{blancas}$ =21000-((76*100) + (1400*4)) =7800

$f.prox\_cas_{negras}$ =21000-((0*100) + (1400*14))=0 (si no hay piezas sobre el tablero, la función es igual a 0)

Según el cálculo de los factores de la función de evaluación se aplica la fórmula para obtener el valor de este factor:

$f.prox=(f.prox\_cas_{blancas}- f.prox\_cas_{negras}) / (f.prox\_cas_{blancas}+ f.prox\_cas_{negras}+1)$

f.prox= 0,999

$f.num\_piez_{blancas}$=10

$f.num\_piez_{negras}$= 0

Aplicamos la fórmula anterior:

$f.num\_piez=(f.num\_piez_{blancas}- f.num\_piz_{negras}) / (f.num\_piez_{blancas}+ f.num\_piez_{negras}+1)$

f.num_piez= 0,909

Por tanto, **func.ev**=200*0.999 + 200*0.909= **381,6**

Situación 2: Casilla extremo lateral de la parte superior del tablero

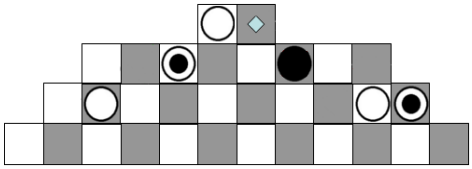


Figura 4. 11

f.prox_cas $_{blancas}$ =21000-((7*100) + (1400*9)) =7700

f.prox_cas $_{negras}$ =21000-((14*100) + (1400*13))=1400

Según el cálculo de los factores de la función de evaluación se aplica la fórmula para obtener el valor de este factor:

f.prox=(f.prox_cas $_{blancas}$- f.prox_cas $_{negras}$) / (f.prox_cas $_{blancas}$+ f.prox_cas $_{negras}$+1)

f.prox= 0.692

f.num_piez $_{blancas}$=5

f.num_piez $_{negras}$= 1

Aplicamos la fórmula anterior:

f.num_piez=(f.num_piez$_{blancas}$- f.num_piz$_{negras}$) / (f.num_piez$_{blancas}$+ f.num_piez$_{negras}$+1)

f.num_piez= 0,571

Por tanto, **func.ev**=200*0.692 + 200*0.571= **252,6**

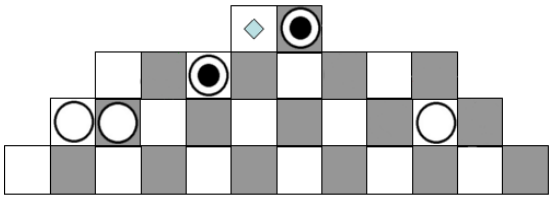


Figura 4. 12

f.prox_cas $_{blancas}$ =21000-((7*100) + (1400*9)) =7700

f.prox_cas $_{negras}$ =21000-((0*100) + (1400*14))= 0 (si no hay piezas sobre el tablero, la función es igual a 0)

Según el cálculo de los factores de la función de evaluación se aplica la fórmula para obtener el valor de este factor:

f.prox=(f.prox_cas $_{blancas}$- f.prox_cas $_{negras}$) / (f.prox_cas $_{blancas}$+ f.prox_cas $_{negras}$+1)

f.prox= 0.999

f.num_piez $_{blancas}$=5

f.num_piez $_{negras}$= 0

Aplicamos la fórmula anterior:

f.num_piez=(f.num_piez$_{blancas}$- f.num_piz$_{negras}$) / (f.num_piez$_{blancas}$+ f.num_piez$_{negras}$+1)

f.num_piez= 0,833

Por tanto, **func.ev**=200*0.999 + 200*0.833= **366,4**

Situación 3: Posición central del tablero, en la parte más alejada al castillo oponente

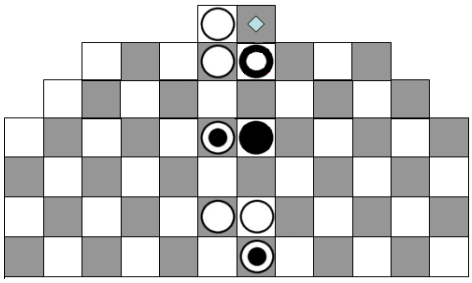


Figura 4. 13

f.prox_cas $_{blancas}$ =21000-((20*100) + (1400*8)) =7800

f.prox_cas $_{negras}$ =21000-((26*100) + (1400*12))= 1600

Según el cálculo de los factores de la función de evaluación se aplica la fórmula para obtener el valor de este factor:

f.prox=(f.prox_cas $_{blancas}$- f.prox_cas $_{negras}$) / (f.prox_cas $_{blancas}$+ f.prox_cas $_{negras}$+1)

f.prox= 0.659

f.num_piez $_{blancas}$=6

f.num_piez $_{negras}$= 2

Aplicamos la fórmula anterior:

f.num_piez=(f.num_piez$_{blancas}$- f.num_piz$_{negras}$) / (f.num_piez$_{blancas}$+ f.num_piez$_{negras}$+1)

f.num_piez= 0,444

Por tanto, func.ev=200*0.659 + 200*0.444= **220,6**

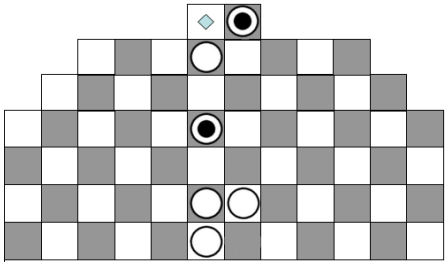


Figura 4. 14

f.prox_cas $_{blancas}$ =21000-((20*100) + (1400*8)) =7800

f.prox_cas $_{negras}$ =21000-((0*100) + (1400*14))= 0 (si no hay piezas sobre el tablero, la función es igual a 0)

Según el cálculo de los factores de la función de evaluación se aplica la fórmula para obtener el valor de este factor:

$$f.prox=(f.prox\_cas_{blancas}- f.prox\_cas_{negras}) / (f.prox\_cas_{blancas}+ f.prox\_cas_{negras}+1)$$

f.prox= 0.999

f.num_piez $_{blancas}$=6

f.num_piez $_{negras}$= 0

Aplicamos la fórmula anterior:

$$f.num\_piez=(f.num\_piez_{blancas}- f.num\_piz_{negras}) / (f.num\_piez_{blancas}+ f.num\_piez_{negras}+1)$$

f.num_piez= 0,857

Por tanto, func.ev=200*0.999 + 200*0.857= **371,2**

La tabla resumen de los resultados es la siguiente:

	FUNCIÓN DE EVALUACION	
SITUACIÓN	Mueve **HOMBRE**	Mueve **CABALLERO**
Situación 1	102,6	381,6
Situación 2	252,6	366,4
Situación 3	220,6	371,2

Tabla 4.24

Observamos con estas tres situaciones que en el primer caso, la relación entre ambos números, es de 3,71, en el segundo caso la relación entre ambos casos, es de 1,45, y en el tercer caso es de 1,68. Teniendo en cuenta que las condiciones de proximidad al castillo eran las mismas para hombres que para caballeros en las tres situaciones, vemos que el número de piezas enemigas capturadas toma un importante papel en el juego. Vemos que en el primer caso, con 6 piezas capturadas, la relación es mucho mayor que en los otros dos casos, con dos y una captura. Así que, llegamos a la conclusión de que el movimiento de carga del caballero, con la que una pieza puede

avanzar y capturar a piezas enemigas, le proporciona al caballero una ventaja cuantitativa en el juego.

Por tanto, la relación entre ambos podría tener esta forma

$$C=H + (capturas)*k$$

El caballero supera el valor del hombre en el valor de una captura tantas capturas como haya realizado.
Obtenemos el valor de k aplicando los resultados de la tabla anterior y obtenemos que

SITUACIÓN	C=H + (capturas)*k	k
Situación 1	381,6=102,6+6k	46,5
Situación 2	366,4=252,6+k	113,8
Situación 3	371,2= 220,6+2k	75,3

Tabla 4.25

De modo que, se aproxima que el valor relativo entre un hombre y una pieza de caballero es de

$$K\approx 75$$

El resultado es una media de los tres valores. A mayor número de capturas en juego entran a formar parte en mayor medida otra serie de factores, que también influyen en la fórmula anterior, de ahí los diferentes valores obtenidos de k. Nos aproximamos de esta manera al valor de k en la situación 3, que tiene un número bajo de capturas.

De modo que, queda demostrado que una pieza de hombre, supone un 75% de la valía de una pieza de caballero.

$$\mathbf{H=0,75C}$$

4.4.2. Mapas de influencia

Cuando una pieza se aproxima al castillo oponente para tomarlo, las piezas encargadas de la defensa del mismo deben tomar partida. Se pretende alcanzar la situación de *bloqueo*, consistente en que la pieza que ataca queda detenida ante el

movimiento de avance, sólo puede retroceder. En caso de que la pieza atacante se encuentre en una fila más cercana al castillo que las que defienden, no hay posibilidad de *bloqueo*.

4.4.2.1 El alcance de los mapas de influencia

En un principio, el análisis de las situaciones para determinar los mapas de influencia se estableció como las 20 casillas correspondientes a las dos primeras filas más próximas a un castillo.

Tras diversos análisis y teniendo en cuenta que se descartan jugadas en las que fichas que se encontraban fuera de ese rango tomaban un importante papel en la resolución del movimiento (victoria o derrota), se modificó el rango de estos mapas, manteniendo las 20 casillas, y pasando a ser tres filas las que entran en juego, tomando mas en importancia el ataque frontal que el proveniente de los flancos en los que las jugadas toman unas características ya recogidas en otras tablas, tal y como se indica en la figura 4.15.

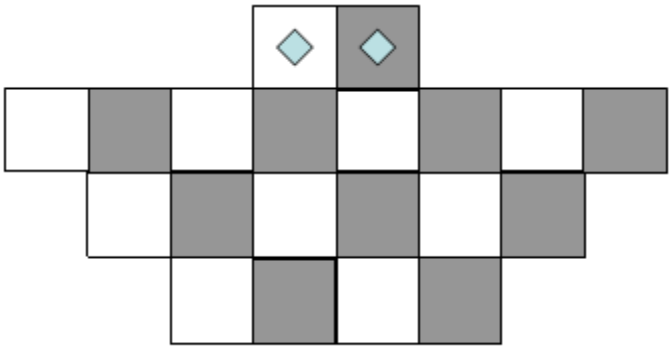


Figura 4. 15

Si hablamos de aquellos ataques en los cuales un caballero forma parte de la ofensiva, tomando parte de un combo junto con un hombre en el cual avanzaría tres casillas en lugar de una, (pasando por encima de su pieza), estaríamos descartando una parte importante del análisis de la situación si solo tomáramos las dos filas más próximas al castillo, ya que de este modo, dejaríamos fuera al atacante caballero, y tendríamos solamente en cuenta al hombre que tiene delante, como si solo una pieza se estuviera aproximando al castillo.

Mas interesa estudiar las situaciones cuanto más cerca estén de las casillas castillo. Descartar los lados de las dos primeras filas supone no tener en cuenta casillas

que están a más distancia del objetivo y generan menos peligro para la defensa, porque van a tardar más en acercarse. Como se puede observar en la figura 4.16, la captura de las piezas de las posiciones centrales, no se tiene en cuenta analizando las dos primeras, filas, pero queda recogido con la eliminación de los extremos inferiores y añadiendo casillas centrales.

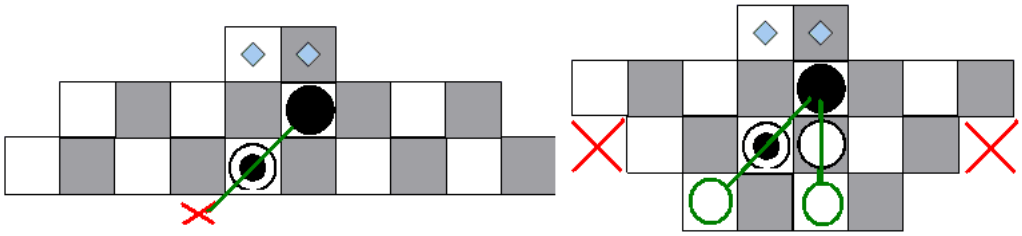


Figura 4. 16

Si la pieza atacante se encuentra al menos una fila más cercana al castillo que el defensor:

Teniendo en cuenta de que es el turno de blancas, es imposible el bloqueo, sea cual sea la posición de las piezas. Si la pieza que pretende conquistar el castillo no comete ningún error en su ataque, el sitio está asegurado (figura 4.17).

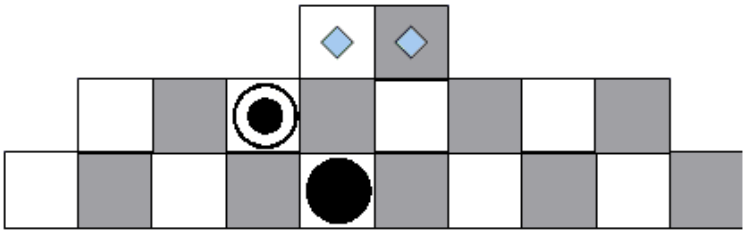


Figura 4. 17

Si la pieza atacante se encuentra en la misma fila que la pieza que defiende el castillo:

Siempre a una distancia de 2 casillas entre ellas, ya que con 1, el atacante *salta* sobre la pieza de su oponente. Como se puede observar en la figura 4.18, y teniendo en cuenta que es el turno de blancas, la pieza de color negro que defiende, tiene la posibilidad de bloquear la movilidad del contrario, a no ser que éste entregue su pieza, haciéndole retroceder y adquiriendo así cierta ventaja. Esta situación se produce si la pieza que ataca se encuentra en las casillas del borde del tablero (1, 2, 7, 8, 9 y 18), marcadas con un recuadro rojo en la figura 4.18

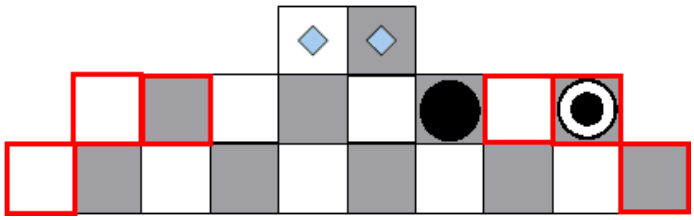


Figura 4. 18

En caso de que el defensor diste más de 2 casillas del usurpador del castillo, como se observa en la figura 4.19, debe esperar a que avance y se sitúe a 2 casillas, porque si no, la pieza negra que defiende es vulnerable a una captura.

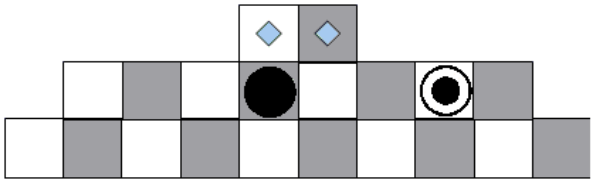


Figura 4. 19

- **Si la pieza atacante se encontrara en las casillas centrales de la zona de influencia:**

  Aquellas que no estuvieran junto a los bordes, de la 10 a la 17. El defensor, no podría bloquear su movimiento, ya que, sólo en los bordes, se resta su movilidad, obligándola a permanecer parada, lanzarse a ser capturada o retroceder. De este modo, en estas casillas centrales de la segunda fila, el atacante tiene la posibilidad de:

  -Escapar retrocediendo filas (lo que le sitúa en desventaja al ataque)

  -Escapar retrocediendo columnas (en sentido contrario a la defensa)

  -Escapar avanzando filas (lo que le proporciona ventaja en el ataque)

- **Si la pieza atacante se encuentra en las casillas 3, 4, 5 y 6**, la entrada al castillo del atacante es inminente; sea cual sea la posición de la defensa.

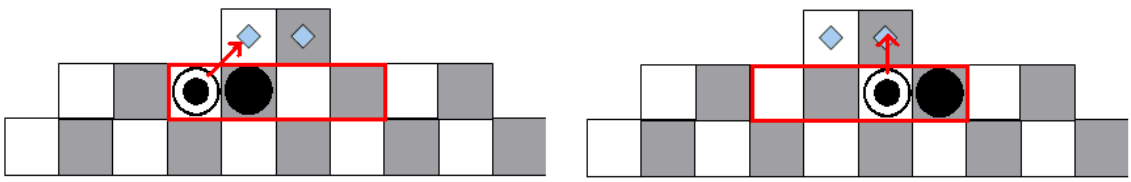


Figura 4. 20

Como se puede observar en la figura 4.20, es imposible el bloqueo de una pieza en estas casillas, si es su turno de movimiento

- **Si las piezas están entre la tercera fila y el centro del tablero** se pueden producir infinidad de situaciones en las que tanto el atacante como el defensor, pueden optar por diversas opciones. De este modo, para asegurarse el bloqueo, el defensor debe manejar la situación para dirigir al atacante hacia las situaciones anteriores en las que puede frenar su intrusión al castillo.

Si la pieza defensora se encuentra entre el ataque y el objetivo (el castillo):

En esta situación, prácticamente en cualquier posición, hay desventaja para la pieza atacante, ya que se puede frenar su avance. Si se deja avanzar filas a las piezas atacantes, dejando de estar entre ellas y el castillo, nos situamos en el primer caso explicado, Figura 4.16, y no se podrá llevar a cabo el bloqueo.

El atacante en las casillas laterales del tablero:

En este caso, la pieza atacante, no se puede bloquear. Al no tener casillas vacías a ambos lados, sobre las que pueda caer el rival al saltar, está resguardado ante ser comida, y puede avanzar sin problema. Estas casillas son las correspondientes a los flancos, y seleccionadas con un recuadro rojo en la figura 4.21.

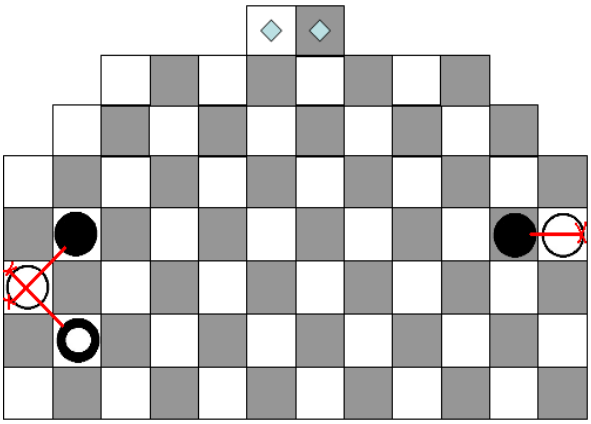


Figura 4. 21

Pero para poder salvar esta situación, se puede recurrir a dos estrategias:

- **Obligar a separar a las piezas de los bordes del tablero;** cuando el atacante llega a las casillas 19 ó 30, marcadas en la figura 4.21 con un recuadro rojo, se amenaza situando a la pieza que defiende en las casillas 1 u 8 respectivamente, marcadas en la figura 4.22 con un recuadro verde, imposibilitando al atacante a seguir avanzando pegada al lateral y obligándole a salir al centro del tablero (dejándole así más vulnerable).

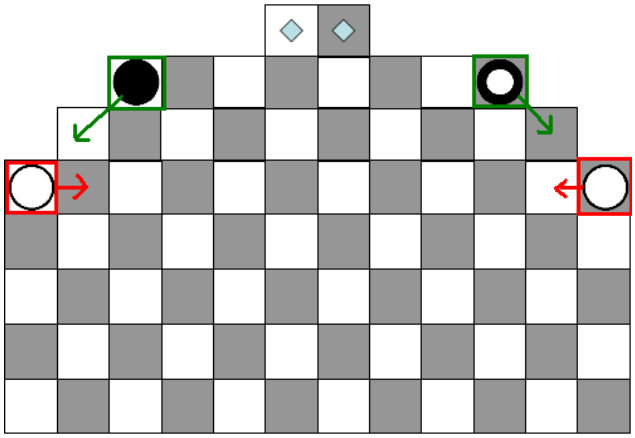


Figura 4. 22

- **Situar la pieza defensiva en la misma columna que la pieza atacante;** de este modo, estando en la misma línea vertical, y dejando una fila en medio de ambas, imposibilita el avance de la pieza que ataca el castillo. Este método también se puede aplicar a las piezas que atacan en la parte central del tablero, como se observa en la figura 4.23.

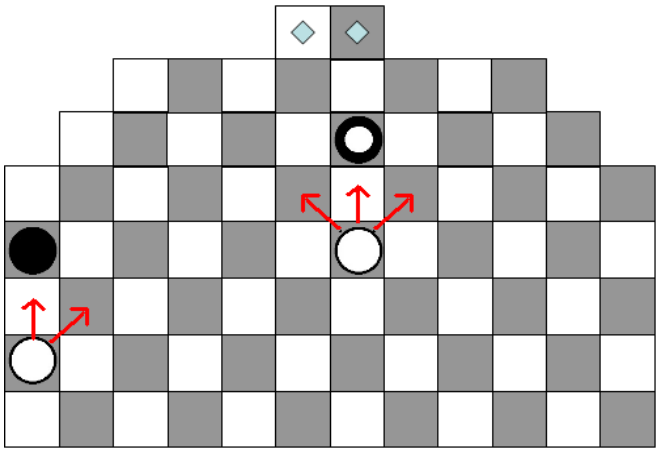


Figura 4. 23

4.5. Función de evaluación

La elección de una buena función de evaluación es uno de los factores primordiales para que el programa seleccione las jugadas adecuadas durante una partida. Si en cualquier instante del juego, el programa es capaz de evaluar, la situación real y actual de la partida, y determinar a favor de cuál de los jugadores se inclina el resultado, y del mismo modo, para cada uno de los posibles movimientos dentro de un turno, siempre escogerá aquel que sea más beneficioso y descartará todos aquellos que lo sean menos.

Una función de evaluación viene determinada por una serie de características que se reflejan en el tablero en el momento de la partida. Es difícil que un solo factor o característica pueda indicar la ventaja de un jugador sobre otro, sino que en la mayoría de los casos, son un número elevado de factores que en unas ocasiones determinan que un jugador está por delante, así como en otras, en el mismo momento, se sitúa por detrás de su contrincante.

En la búsqueda de la función de evaluación más adecuada es importante incluir una combinación de todas aquellas características que son importantes en el desarrollo del juego y que dan ventaja a los jugadores, ya que, hay numerosos factores que influyen en el desarrollo del juego, pero no son lo suficientemente significativos para la toma de decisiones.

Una función que incluya todas aquellas características y que consiga determinar el peso o importancia (w) que estas tienen para la partida, incluso, pudiendo variar este a lo largo del tiempo de juego (t), es muy importante para el desarrollo y creación de un programa que funcione eficientemente

La fórmula de la función de evaluación tiene la siguiente forma:

$$F(e)=\sum^{n}_{i=0} (w_i).(t_i).(f_i)$$

Donde f_i son cada una de las funciones (o factores) que componen la función de evaluación global.

A continuación pasaremos a explicar cada uno de los componentes de esta función tan importante para el programa.

4.5.1. Factores de la función de evaluación (f_i):

A continuación vemos una descripción de todos los factores de la función de evaluación.

4.5.1.1. Número de piezas (f_1)

En primer lugar, uno de los factores más importantes a la hora de evaluar la situación del juego, es el número de piezas que intervienen en él. En caso de que un jugador haya perdido una gran parte de sus piezas, obviamente, se encuentra en desventaja frente a su oponente, el cual, tiene más posibilidades de cubrir sus ataques y de defender su castillo.

El número de piezas por jugador es de 14; por tanto, el valor de f_1 corresponde con el número de piezas que posee el jugador, que, cuanto mayor sea, influirá más positivamente en el valor de su función de evaluación. El valor mínimo que va a devolver esta función es 0 y el máximo 14.

Un pequeño ejemplo de cómo contabiliza la función sería el siguiente, reflejado en la figura 4.24

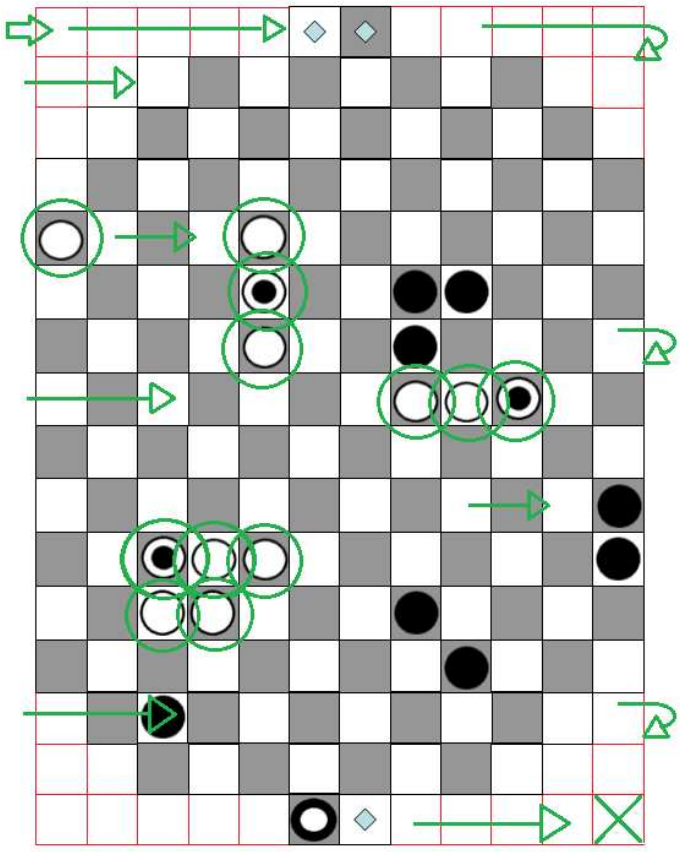


Figura 4. 24

Los resultados:

Número de piezas blancas= 12

Número de piezas negras = 9

Se puede consultar el código del método en el apartado de apéndices; 8.3.1.

4.5.1.2. Seguridad (f_2)

La seguridad como factor de la función de evaluación representa la situación de las piezas del color que deseemos analizar, utilizando la combinación de dos agentes influyentes, estos son, el número de piezas que defienden el castillo y el número de piezas enemigas que atacan mi castillo. Viene a querer decir cuán seguras están las piezas, respecto a la situación de las enemigas, dependiendo del número de ellas y su situación. Se encarga de, dividiendo el tablero en dos zonas, correspondientes al territorio de las piezas blancas (filas de la 8 a la 15) y el territorio de las piezas negras (filas de la 0 a la 7), evalúan, en caso de piezas blancas, el número de piezas de ambos colores en la zona de su castillo, y las relacionan de modo que obtenernos un valor que posee un rango entre -0.5 y 0.5 y nos indica si poseemos una buena o mala seguridad de las piezas a estudio.

Este factor es un poco diferente al resto de los que componen la función de evaluación, ya que aquí se evalúan dos agentes por separado, para luego relacionarlos mediante la siguiente fórmula:

Seguridad= (bb-nb) / bb+nb+1

Siendo bb el número de piezas blancas en territorio blanco y nb piezas negras en territorio blanco.

Del mismo modo, en caso de análisis de las piezas negras, la formula sería la misma, pero variaría la zona de estudio, quedando así:

Seguridad= (nn-bn) / nn+bn+1

Siendo en este caso nn el número de piezas negras en territorio negro y bn, las piezas blancas en territorio negro.

Estas dos fórmulas se pueden resumir en un caso general, de la siguiente manera:

Seguridad= (defensa-ataque) / (defensa+ataque+1)

El valor de salida de este factor, está comprendido entre -1 y 1. La figura 4.25 recoge un ejemplo de funcionamiento de este factor sobre el tablero.

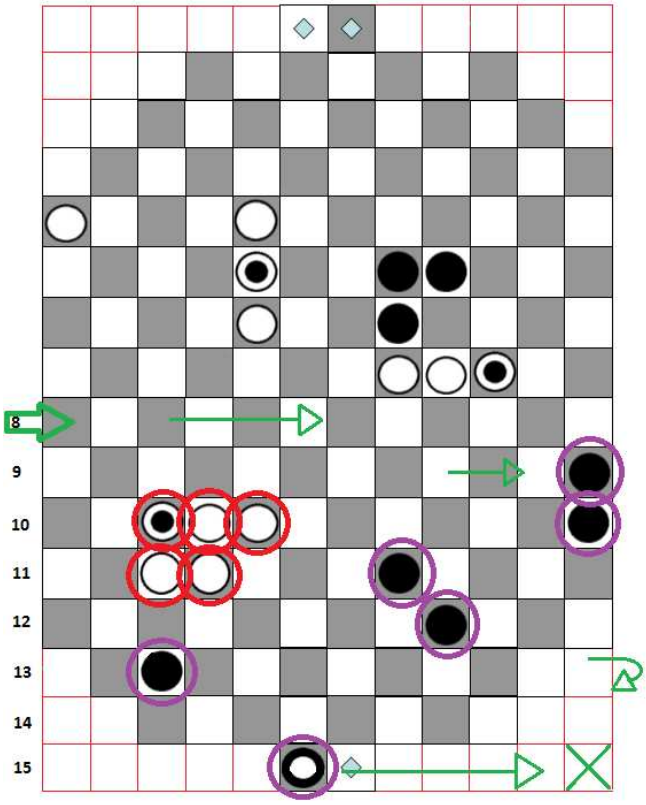


Figura 4. 25

Los resultados.

Seguridad blancas= -0,083;

Seguridad negras=- 0,36;

La seguridad de las piezas blancas, no está a la altura del ataque del contrincante, pero las piezas negras son vulnerables al ataque blanco también. Si comparamos ambas seguridades, estas últimas están peor protegidas que las blancas.

Se puede consultar el código del método en el apartado de apéndices; 8.3.2.

4.5.1.3. Número de caballeros (f_3)

Como se ha comprobado en apartados anteriores, el caballero posee un valor canónico superior al de las piezas hombre, por las ventajas que supone poder llevar a cabo el movimiento exclusivo de *galope*, no permitido en las piezas de hombre. De este modo, aquel jugador que posea un mayor número de piezas caballero entre sus piezas en juego, poseerá mayor ventaja que su oponente.

Igualmente en la figura 4.26, se muestra un ejemplo del resultado obtenido de la evaluación de este factor sobre el tablero que hemos tomado anteriormente.

El valor que va a devolver este factor está comprendido entre 0 y 4. Se puede consultar el código del método en el apartado de apéndices; 8.3.3.

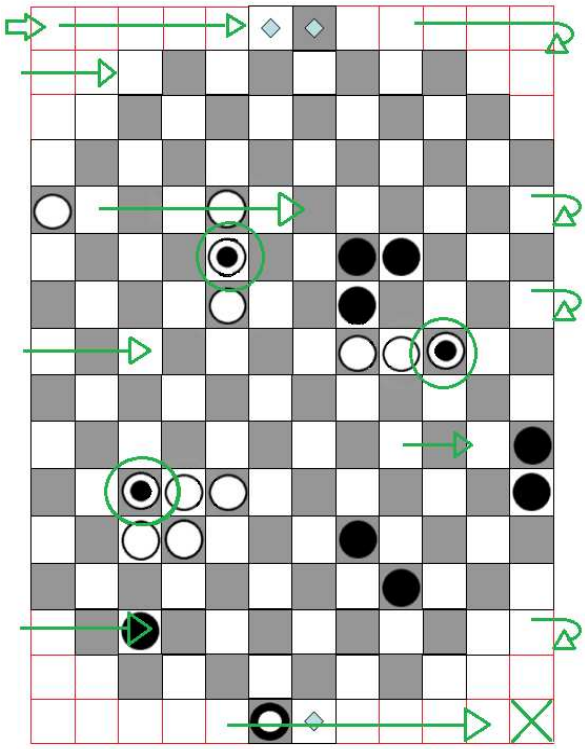


Figura 4. 26

Los resultados:

Número de caballeros blancos= 3

Número de caballeros negros= 1

4.5.1.4. Proximidad al castillo (f_4)

Este factor fue incluido con posterioridad al conjunto de la función de evaluación, no formaba parte del grupo inicial de factores.

Se trata de un concepto muy sencillo pero a la vez sumamente importante en el desarrollo del juego, como lo pueden ser el número de piezas o los castillos ocupados. Una de las situaciones que nos puede llevar a la victoria en el juego de CAMELOT es la ocupación de los castillos (la otra es dejar al contrincante con menos de dos piezas), por tanto, cuanto más cerca este la masa de tus piezas del castillo que pretendes ocupar, más probabilidad de alcanzar la victoria tendrás. Si este factor posee el peso suficiente, cuando una pieza se mueva en dirección al castillo, esto, proporcionara más valía a la

función de evaluación del tablero en el análisis que se lleva a cabo en el árbol de búsqueda. Por tanto, en caso de que no se pueda realizar una captura, o un movimiento evidentemente favorable, se escogerá desplazar la pieza elegida hacia el castillo enemigo, antes que cualquier otra dirección.

El valor máximo de la proximidad

Se ha obtenido el valor límite de 21000 para esta constante de la siguiente manera. En el caso más pesimista en cuanto a distancia al castillo enemigo, la disposición de las piezas sería la indicada en la figura 4.27

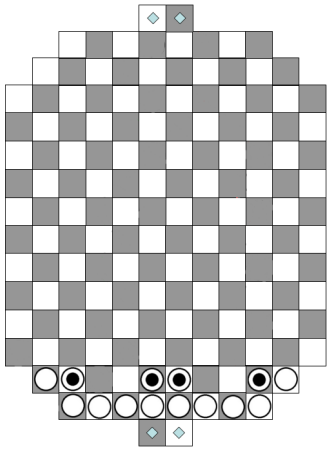


Figura 4. 27

En cuyo caso, la distancia total de todas las piezas al castillo oponente sumaría 190, multiplicado por 100 para darle amplitud al resultado, 19000.

En el caso más optimista, en el que todas las piezas estén lo más cerca posible del castillo, teniendo en cuenta que son 14, se tienen que distribuir por la primera y segunda fila del tablero, finalmente sumando 20 casillas de distancia al castillo, como se observa en la figura 4.28.

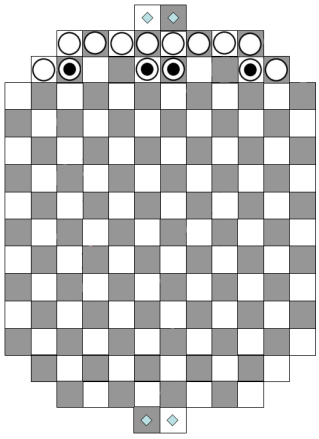


Figura 4. 28

De modo que, como se descuenta 100 de la constante inicial por cada casilla de distancia, nunca podrían alcanzar la puntuación máxima de 19000, porque aun en la mejor de las situaciones, están a 20 casillas del objetivo. Por tanto, se establece como máximo 21000, para que 19000 sea la puntuación máxima a obtener, solamente posible si todas sus piezas sobre el tablero se encontraran en la primera fila, delante del castillo enemigo. El rango del valor que devuelve esta función es de [0, 19000].

El comportamiento de este factor sobre el tablero ejemplo anteriormente utilizado, se muestra en la figura 4.29.

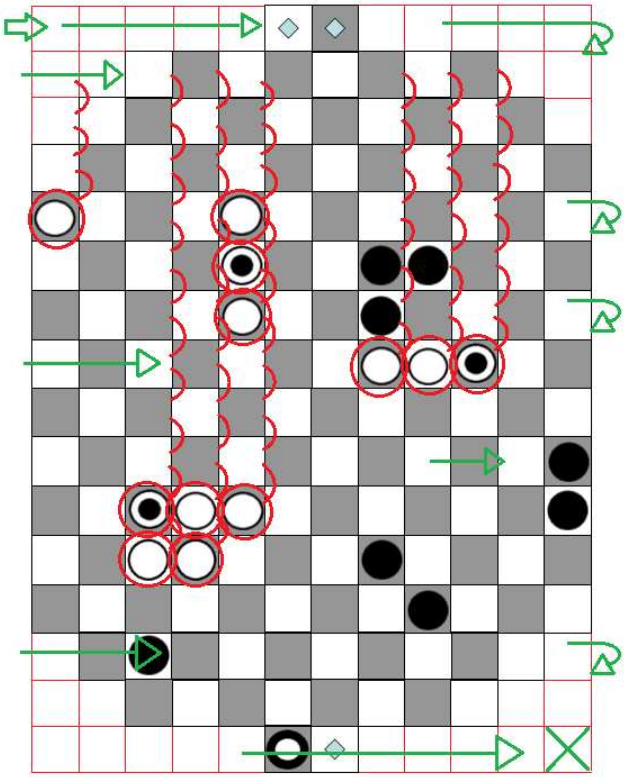


Figura 4. 29

Los resultados:

Proximidad al castillo blancas= 9000;

Proximidad al castillo negras= 9100;

Se puede consultar el código del método en el apartado de apéndices; 8.3.4.

4.5.1.5. Proximidad a los flancos (f_5)

Aquel jugador que tenga situadas sus piezas en la parte central del tablero se encuentra en desventaja frente a su oponente, ya que tiene más expuestas sus piezas al ataque del contrario, por las partes superior, inferior y laterales. Si los hombres y los

caballeros se sitúan de manera estratégica en los flancos del tablero, imposibilita determinados movimientos de la ofensiva a ser capturados. Aquel jugador que posea piezas en los laterales se encuentra en una posición de ventaja.

Como ya se explicó antes brevemente en el apartado 2.7.1 de los mapas de influencia, y como se indica en la figura 4.30, observamos que:

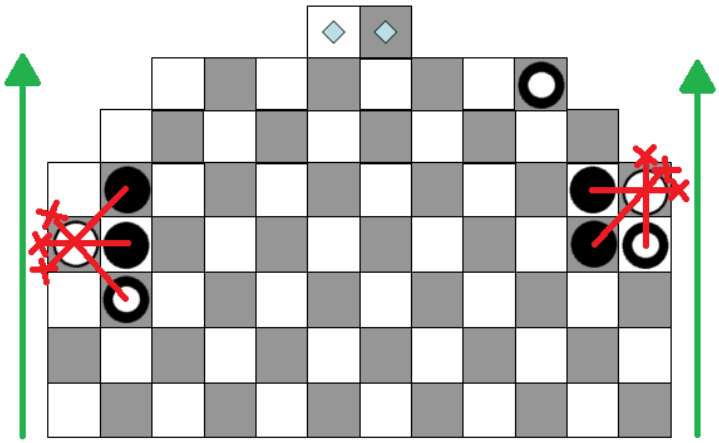


Figura 4. 30

Cualquier pieza que avance (en este caso) por los laterales del tablero, presenta una clara ventaja, ya que reduce de ocho a dos las posibles capturas que su enemigo le puede infligir. En este ejemplo, salvo que se acerque el jugador defendiendo, a la pieza blanca de manera vertical por arriba o por abajo, el resto de movimientos de captura mientras se desplace por el flanco son imposibles (las capturas en diagonal y la horizontal) como se puede observar en el caso a la izquierda en el tablero ejemplo.
Si nos fijamos en la situación de la parte derecha del tablero, observamos que al llegar a la cuarta fila, al no ser posible seguir avanzando de manera vertical y tener que tomar el flanco diagonalmente, la pieza blanca queda algo más expuesta a ser capturada o, en caso de no poder avanzar, puede quedar bloqueada sin posibilidad de escapatoria para el contrario. El valor que devuelve esta función está comprendido entre [0, 12].

Por tanto, las zonas que se analizan en este método se pueden observar en la figura 4.31.

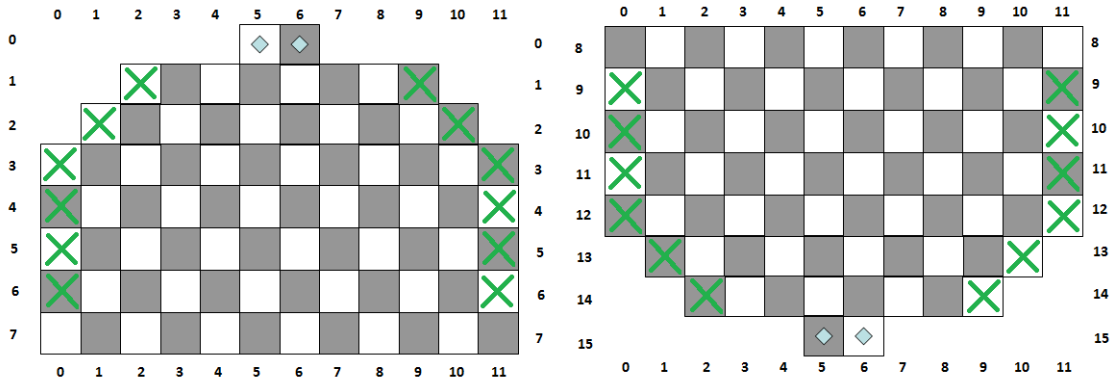


Figura 4. 31

El resultado obtenido del factor *comprueba_flancos* aplicado sobre el tablero ejemplo se muestra en la figura 4.32.

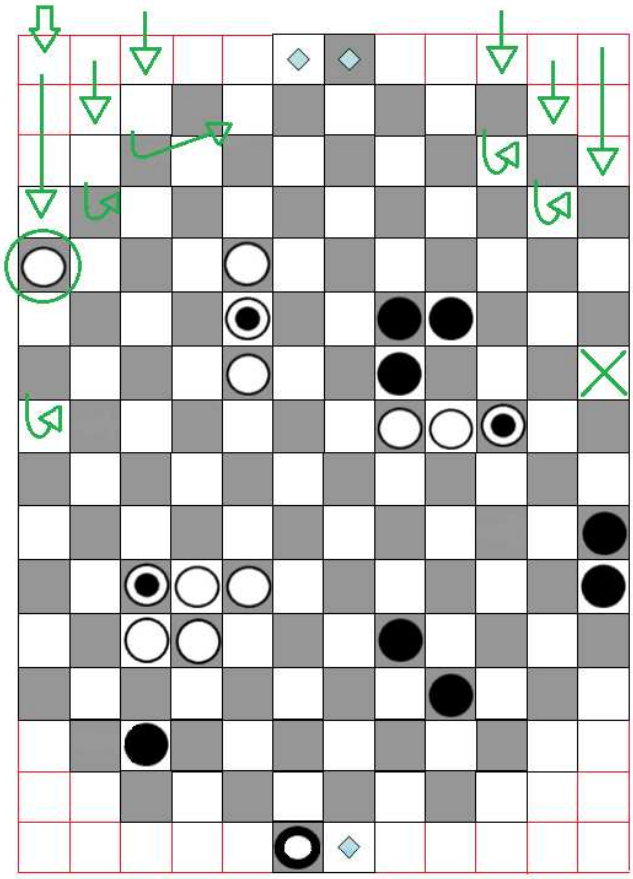


Figura 4. 32

Los resultados:

Flancos blancas=1;

Flancos negras=0;

Se puede consultar el código del método en el apartado de apéndices; 8.3.5.

4.5.1.6. “Pelotones” de piezas (f_6)

Una acumulación de piezas en el tablero del mismo bando implica situación ventajosa ya que no pueden ser capturadas por las piezas enemigas si no poseen huecos entre ellas para llevar a cabo la eliminación. Por tanto, cuanto más desperdigadas estén las piezas, más vulnerables serán al contrario.

Ya no se trata de un conjunto de piezas juntas muy numeroso, es suficiente con que cuatro piezas estén juntas del modo que muestra la figura 4.33

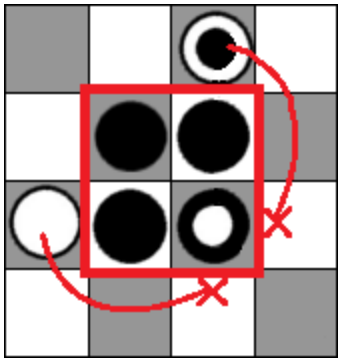


Figura 4. 33

De este modo, como se observa en la figura, no es posible la captura por parte de ninguna de las piezas enemigas desde ninguna dirección en este ejemplo; No se garantiza la supervivencia de las piezas, ya que hay puntos muertos en los que sí es posible la captura, pero, reduce notablemente el ataque contrario.

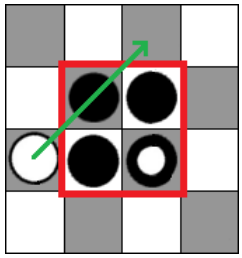


Figura 4. 34. Ejemplo de captura con pelotón

Obviamente, cuanto mayor número de piezas estén juntas, menos posibilidades de que el contrario se "cuele" y salte sobre ellas, pero también es más costoso el mantenimiento de esa posición, dado que al solo poder mover una pieza en cada mano, esta se disgregaría del pelotón y costaría al menos dos manos más conseguir componer el pelotón anterior.

El método comprueba las posiciones colindantes a cada pieza como se indica en la figura 4.35

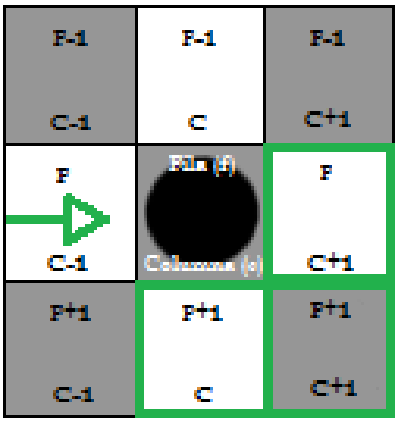


Figura 4. 35

No se analizan el resto de posiciones debido a que, si así se hiciera, y se tuviera en cuenta cada vez que la pieza encontrada tuviera tres fichas contiguas de su mismo color, provocaría probablemente que el mismo pelotón se contabilizara más de una vez, y otorgando de esta manera, más valor del que le corresponde al factor; como observamos en la parte izquierda de la figura 4.36

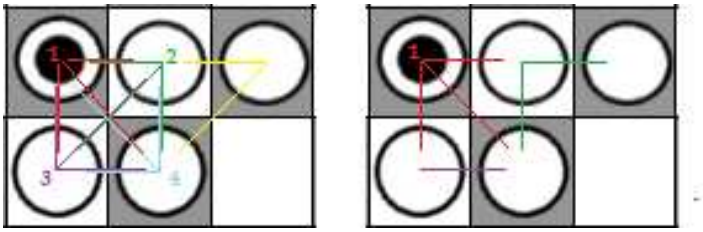


Figura 4. 36

En la parte derecha, se demuestra en cambio como se contabiliza tal y como está establecido en la función.

El valor mínimo que va a devolver este factor es de 0 y el máximo es de 6, porque para que puntúe un pelotón, tienen que formar parte de él 4 piezas, y solo hay 14 en juego, entonces, en la situación óptima en la que todas las piezas se dispusieran juntas, podrían formarse 6 pelotones.

A continuación vemos en la figura 4.37 un ejemplo del funcionamiento del método, sobre el tablero ejemplo utilizado para otros factores

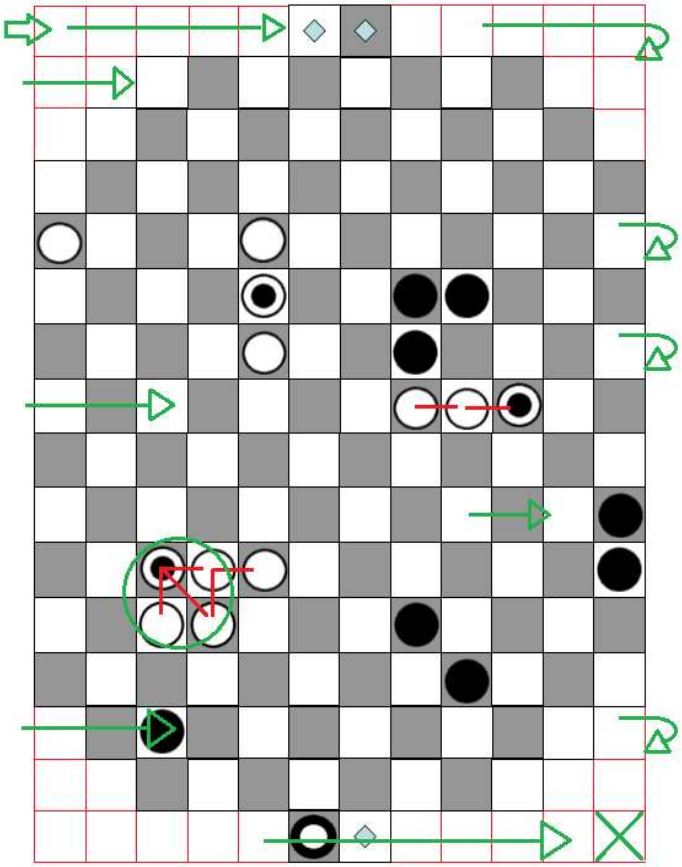


Figura 4. 37

Los resultados:

Pelotones blancas=1;

Pelotones negras= 0;

Se puede consultar el código del método en el apartado de apéndices; 8.3.6.

4.5.1.7. Castillo ya ocupado (f_7)

Se tiene en cuenta que ninguna de las casillas del castillo este ocupada por una pieza contraria. De modo que, aquel jugador que ya haya penetrado con una pieza en el castillo de su oponente ya tendrá una ventaja sobre él, ya que posee la mitad del objetivo cumplido

Junto con el número de piezas de juego este es uno de los factores más importantes, ya que, en cualquier partida, en un momento determinado de juego, si nos paramos a observar la situación del tablero, y uno de los dos bandos posee una pieza en

el castillo enemigo, nos decantaremos por decir que la partida evoluciona a favor de este jugador.

Las casillas de los castillos, como ya se ha explicado antes, corresponden con los extremos superior e inferior del tablero y es obligación de los jugadores, atacar el castillo del campo oponente, así como defender el suyo. El valor que devuelve esta función está comprendido entre [0,200]. Observamos en la figura 4.38 cómo el factor comprueba los castillos de las piezas blancas.

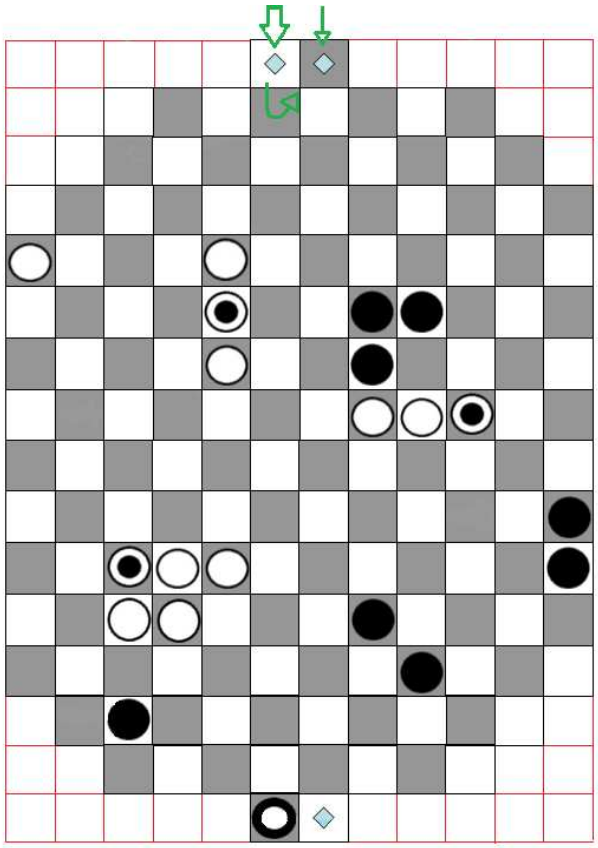


Figura 4. 38

Los resultados:

Castillo ocupado blancas= 0;

Castillo ocupado negras= 1;

Se puede consultar el código del método en el apartado de apéndices; 8.3.7.

4.5.1.8. "Concentración" de piezas (f_8)

Se calcula el número de piezas que se encuentran juntas en territorio enemigo. Esto supone una situación ventajosa ya que su proximidad al castillo oponente es inferior y aumenta la ventaja cuanto mayor sea el número de atacantes que se encuentren.

Es una combinación de calcular la proximidad al castillo enemigo de las piezas (lo que supone una ventaja notable, porque cuanto más cerca tengas tus piezas del castillo objetivo, más posibilidades de conquistarlo) junto con el factor de pelotones de piezas, lo que convierte a las piezas, cuanto más cerca y más juntas, en mucho más competitivas en la partida.

El método de *concentración* hace distinción en cuanto al color de las piezas que queramos evaluar, esto es debido a que, tratándose por ejemplo de las piezas blancas, analizamos solamente, la parte superior del tablero, debido a que estamos valorando nada más que las piezas se encuentren en territorio enemigo, aquellas que estén por debajo de la mitad del tablero, no son piezas competitivas. A aquellas que se sitúen por encima de la fila 7 del tablero, serán puntuadas en el método del cálculo de la concentración.

Del mismo modo, en caso de análisis de las piezas negras, la zona del tablero competitiva es la inferior, incluyendo todas las filas por debajo de la 12, donde las piezas negras, tienen más valor para la función de evaluación

Por cada pieza, se evalúan las casillas contiguas derecha e inferior (indicado con recuadros verdes y azules en la figura 4.39), como en el caso de los pelotones, siguiendo el sentido de la búsqueda.

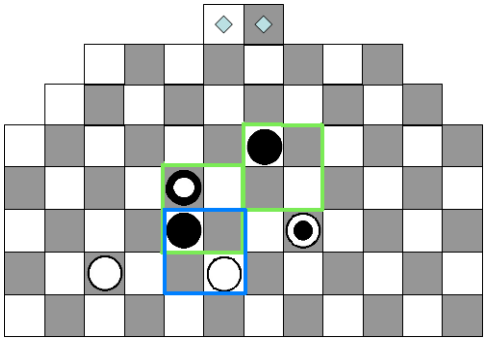


Figura 4. 39

Una vez situados en cada una de estas casillas que denominaremos *cuadro*, se evalúan el lado derecho contiguo, el inferior, y el inferior derecho, como se muestra en la figura 4.40; como en el caso de análisis de los pelotones.

Figura 4. 40

Por cada pieza que encontremos en estas tres casillas, incrementamos en 1 un contador inicializado a 0 al comienzo de la ejecución, con nombre k (vuelve a valor 0 en cada cuadro); tras esto, hacemos un último análisis, añadimos más valía a estas piezas concentradas cuanto más cerca se encuentran del castillo objetivo a través de la siguiente fórmula:

$$C= c + (6 - i) *k$$

La i corresponde con la fila en la que nos encontramos, y se le resta a 6, que son el número máximo de filas que evaluamos; cuanto menor sea el número de i, mejor resultado porque nos encontramos en las filas más próximas al castillo. Este número se multiplica por k, que es el número de piezas encontradas en el *cuadro*, y por tanto, a modo de pelotón, lo que le da mayor valor cuanto más piezas haya (como máximo 4). El cálculo para las piezas negras se hace de forma análoga. El valor mínimo que devuelve esta función es 0 y el máximo 168, ocasión más optimista en la que todas las piezas se encuentran conjuntas en las dos primeras filas delante del castillo

Si aplicamos esta función sobre el ejemplo de tablero, que muestra la figura 4.41, y que hemos tomado para el resto de factores, el resultado sería el siguiente:

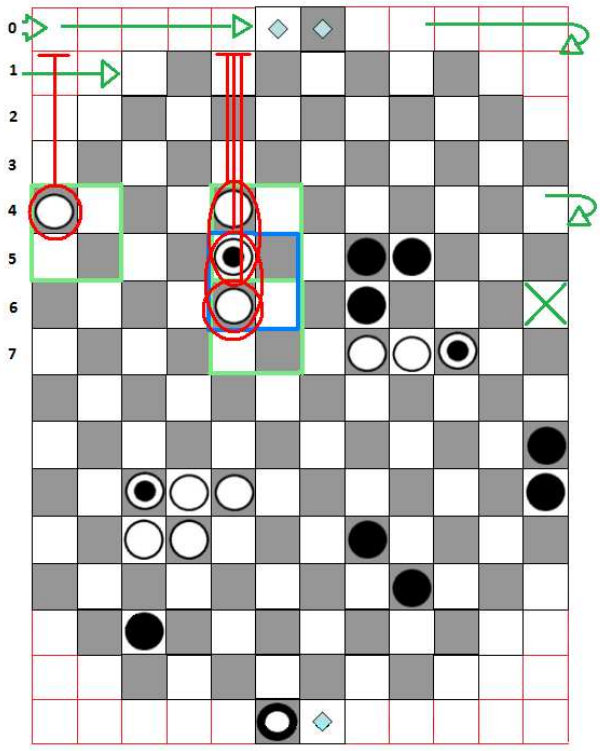


Figura 4. 41

Los resultados:

Concentración blancas= 8;

Concentración negras= 19

Se puede consultar el código del método en el apartado de apéndices; 8.3.8.

4.5.1.9. Movilidad (f_9)

Se trata del número de posibles movimientos que puede realizar cada una de las piezas que pertenecen a cada jugador. Se suma cada una de ellas y aquel jugador que posea mayor movilidad tiene una ventaja sobre su adversario, ya que dispone de más posibilidades de movimiento.

Si una pieza posee más movilidad, es porque mantiene cierta distancia con su enemigo. Se calcula el número de movimientos por cada pieza, siguiendo el sistema explicado en el capítulo 2.1.2. Esta función posee un rango de [0, 112], cuyo valor máximo se alcanza cuando las 14 piezas en juego, pueden realizar los movimientos en las 8 direcciones. Se muestra el comportamiento del método, sobre al tablero ejemplo anteriormente utilizado, en la figura 4.42.

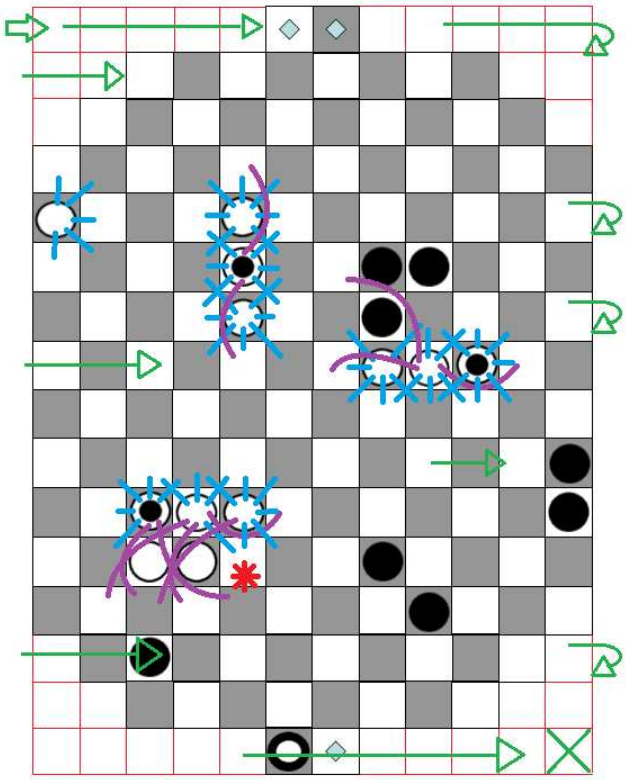


Figura 4. 42 *

En la figura faltan por introducir las líneas de movimiento de las dos últimas piezas blancas, pero no se han incluido por no restarle claridad a la figura. Los resultados:

Movilidad blancas= 85;

Movilidad negras= 57;

Se puede consultar el código del método en el apartado de apéndices; 8.3.9.

4.5.1.10. Número de combos caballero-hombre (f_{10})

La asociación de una pieza hombre con la de un caballero proporciona una ventaja sobre el tablero, ya que, con esta combinación, el avance del caballero por encima del hombre, permite la captura con la carga del caballero y un avance más rápido hacia el castillo enemigo.

La combinación de piezas hombre-hombre, también conlleva un avance rápido, pero según las reglas del juego, el hombre puede saltar por encima de piezas de su mismo color tantas veces como le sea posible, pero nunca sobre una pieza oponente tras haber encadenado saltos. Precisamente el hecho de que el caballero pueda realizar la carga, supone una ventaja competitiva sobre su adversario, ya que puede encadenar saltos y a su vez tantas capturas como le sea posible realizar. Si un caballero tiene una pieza de su color en una casilla contigua, y puede saltar sobre ella (teniendo la casilla destino libre), no solo le permite avanzar, si no también capturar las piezas de su adversario, disminuyendo así su material en juego sobre el tablero

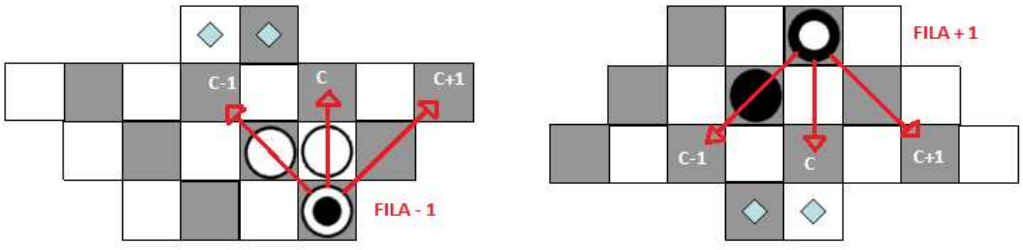


Figura 4. 43

En el código de este método se distingue el análisis de las piezas blancas del de las negras, ya que, hay una pequeña diferencia entre ambas (Figura 4.43). En lugar de comprobar si existen *combos* de caballero-hombre en las o posibles posiciones contiguas al caballero encontrado, solo se tienen en cuenta aquellas que las aproximan

al castillo del oponente (columna+1; columna y columna-1); es decir, no se consideran valiosos aquellos combos en los que la pieza retrocedería hacia su castillo y se descartan también los laterales, que no supondrían un movimiento de avance. El valor mínimo que devuelve esta función es 0 y el máximo es 4, número máximo de caballeros en el tablero

En el tablero ejemplo que hemos tomado en el resto de factores, se muestra el comportamiento del método en la figura 4.44.

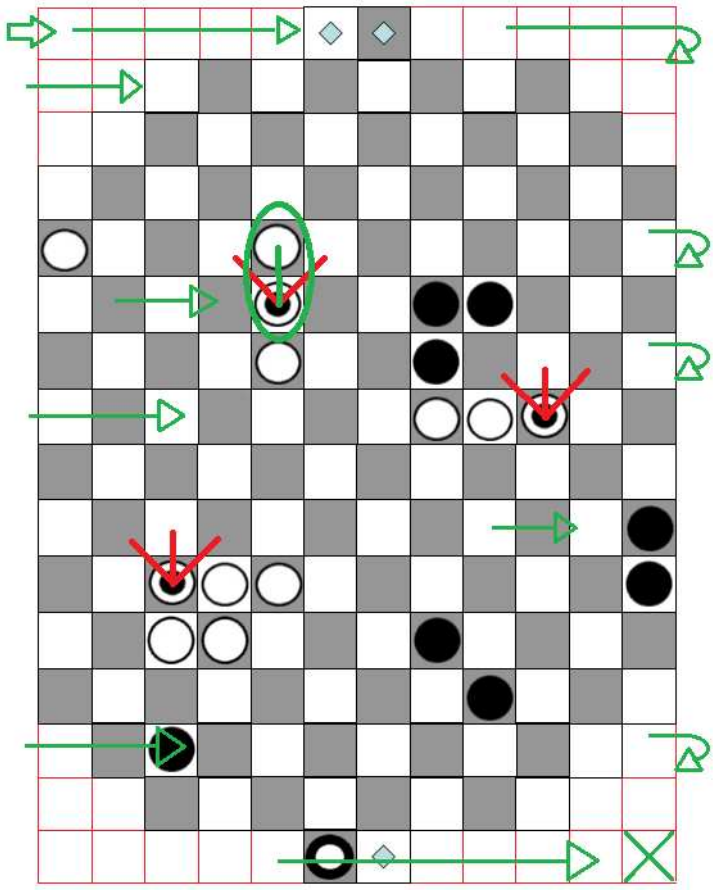


Figura 4. 44

Los resultados:

Número de combos caballero-hombre= 1

Número de combos caballero-hombre=0

Se puede consultar el código del método en el apartado de apéndices; 8.3.10.

4.5.2. Pesos de la función de evaluación (W_i)

Hablando de la importancia que recae sobre los diferentes factores de la función que evalúa el estado de la partida, se llega a la conclusión de que, los diferentes factores tienen diferente influencia sobre el juego. Del mismo modo que, dependiendo del

momento de la partida en el que nos encontremos, influirá de manera distinta. Esta influencia se traduce en un número entero que recibe el nombre de peso y se representa con **W.**

El rango que se ha establecido para determinar los pesos que acompañan a los factores de la función de evaluación está comprendido entre 0 y 200. Si se hubiera escogido un margen demasiado pequeño, no puede darnos la suficiente diferenciación en caso de resultados similares en situaciones parecidas en el desarrollo de la partida; por lo que, escogiendo un rango mayor, permite demostrar distinción, entre ellas.

-Número de piezas (f_1): El peso escogido para este factor es de 200. Es una de los factores con mas pesos, ya que, este, representa uno de los agentes objetivo del juego que es, eliminar todas las piezas de tu adversario hasta que solo quede una, y a su vez, intentar mantener a salvo las tuyas. Cualquier jugador que posea más material sobre el tablero que su adversario, se ve como una notable ventaja.

-Seguridad (f_2): El peso escogido para la seguridad es de 75. Se le asigna un alto valor a este factor porque resulta muy significativo evaluar, cual es la situación de tus piezas respecto a las de tu enemigo en la zona de tu castillo, para saber si se encuentra en ventaja o desventaja.

-Número de caballeros (f_3): El peso escogido para este factor es de 50. Al igual que el número de piezas en juego, el material del que se dispone es importante en la partida, y los caballeros tienen un valor añadido sobre el resto de piezas.

-Proximidad al castillo (f_4): El peso escogido para este factor es de 200. Este es el valor más alto de todos los pesos de los factores ya que, es importante, en la toma de decisión, que BOU se decante por las posiciones que le aproximan al castillo, y a su vez, acercándose a piezas amigas, para evitar ser capturado.

-Proximidad a los flancos (f_5): El peso escogido para este factor es de 50. Es uno de los factores de menos importancia de la función de evaluación, pero a la vez, importante para tener en cuenta, ya que, aquellas piezas que se encuentran en los laterales del tablero, son inmunes a ser comidas por los atacantes y les permiten avanzar con seguridad hasta el castillo.

-Pelotones de piezas (f_6): El peso escogido para este factor es de 75. Es el tercer valor más importante de los pesos, ya que es bastante influyente que las piezas en su avance se vayan disponiendo de modo que se protejan entre ellas.

-Castillo ya ocupado (f_7): El peso escogido para este factor es de 175. Es otro de los factores más importante del juego, ya que, al igual que el número de piezas, es otro de los agentes objetivos del juego, ocupar el castillo. Por tanto, cualquier jugador que ya posea una de sus piezas en el castillo enemigo, presenta una gran ventaja frente a su enemigo.

-"Concentración" de piezas (f_8): El peso escogido para este factor es de 75. El valor es de carácter medio ya que, no tiene tanta importancia como los de más peso, es simplemente un mecanismo de defensa al que hay que recurrir cuando el resto de factores más importantes no proporcionan una solución inmediata. Sirve para evaluar, la zona de peligro y saber si se está en ventaja o desventaja.

-Movilidad (f_9): Esta función tiene un peso de 75. No posee un valor demasiado alto ya que no es imprescindible, aún así sigue siendo importante que las piezas tengan posibilidades de movimiento, si se encuentran bloqueadas por el contrario o por otras piezas amigas, disminuyen las opciones.

-Combos caballero-hombre (f_{10}): El peso asignado a este factor es de 25. Es el factor con menor importancia de todos los que componen la función. Se tiene en cuenta, porque, de un modo sutil, pero influye en la decisión de juego, hay una ligera ventaja de los jugadores que tienen esta composición de hombres y caballeros.

Es decir las relaciones de importancia de los factores de la función de evaluación (de menor a mayor importancia) se muestran en la figura 4.45.

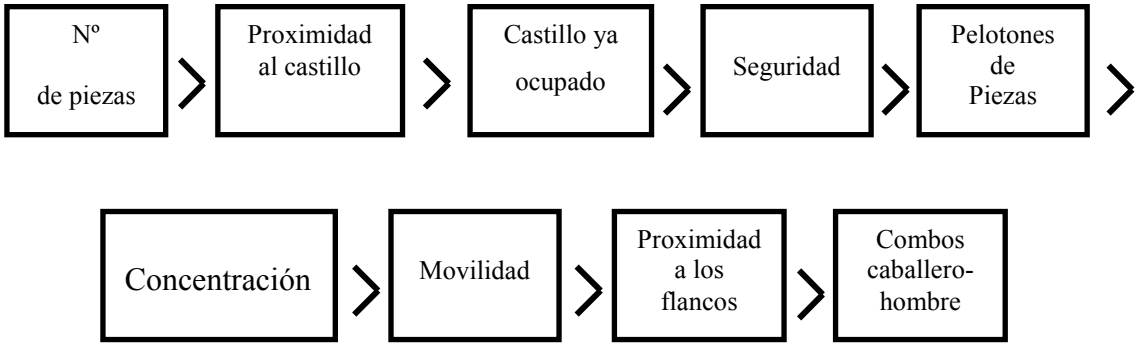


Figura 4.45

Al inicio de la composición de la función de evaluación se establecieron unos valores arbitrarios para estos pesos, y tras la maduración del juego y la práctica en pos

de obtener resultados, se han ido modificando y ajustando para que el funcionamiento y la destreza de BOU fuera lo más óptima posible.
Quedando la fórmula desglosada, así:

$$F(e)=200(t_1).(f_1) + 75(t_2).(f_2) + 50(t_3).(f_3) + 200(t_4).(f_4) +50(t_5).(f_5) +75(t_6).(f_6) +175(t_7).(f_7) +75(t_8).(f_8) + 75(t_9).(f_9)+ 25(t_{10}).(f_{10})$$

La suma de los pesos de todos los factores de la función de evaluación es de 1000, es decir, el valor máximo que puede adquirir la función de evaluación es de 1000, ya que los valores de salida de los factores de manera individual (por cada color) no están comprendidos entre -1 y 1, pero, si lo está el resultado final de cada factor, obtenido de aplicar la siguiente fórmula:

$$f_{i(BOU)} = (f_{i(BOU)} - f_{i\ Oponente}) / (f_{i(BOU)} + f_{i\ Oponente} + 1)$$

(Sin tener en cuenta aquellos que dependen del factor tiempo, que supone una multiplicación más en su valor).

4.5.3. Tiempo en la función de evaluación (t_i): Los Factores "i"

En determinados factores de la función de evaluación el tiempo supone un determinante a la hora de analizar y asignar un valor a la partida.
Según la fórmula que compone la función de evaluación que influye sobre la partida, que tiene este aspecto:

$$F(e)=\sum^{n}_{i=0} (w_i).(t_i).(f_i)$$

Este factor del tiempo posee la representación de t_i e influye sobre todos los factores que componen la función. Pero, no lo hace por igual en todos los factores. Existen tres de ellos en los que la aplicación del tiempo es significativa, en el resto, es irrelevante. Estos son: ***Número de piezas*** y ***Proximidad***.

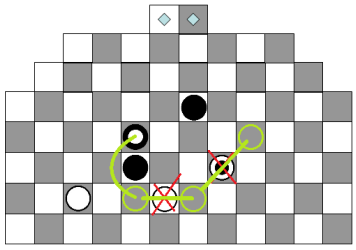
En estos tres casos se considera $\mathbf{t_i}$ un entero que representa el momento de la partida. Coincide con el número de movimiento en el que nos encontremos, es decir, no corresponde con el tiempo en minutos o segundos de la partida, o el tiempo que un jugador lleve invertido en su jugada, si no el número de movimiento que corresponda en cada momento.

Durante el desarrollo del juego, se almacenan dos tipos de número que poseen significados diferentes. Estos son:

-**Ply:** que corresponde con el número de movimientos que se llevan a cabo en la partida, sin tener en cuenta el color del jugador

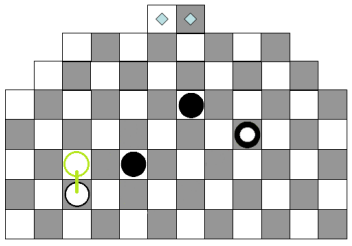
-i: cuenta el número de jugadas que se llevan a cabo en la partida, contando con el movimiento de ambos jugadores.

Es decir, cada **i** contiene dos **ply**, como se aclara en la figura 4.46 a continuación:



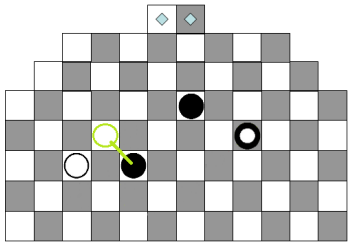
i=1

ply 1- 4 , 4 → 6 , 4 → 6 , 6 → 4 , 8



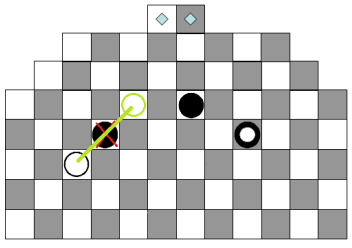
i=1

ply 2- 6 , 2 → 5 , 2



i=2

ply 3- 5, 4 → 4 , 3



i=2

ply 4- 5 , 2 → 3 , 4

Figura 4.46.

Es decir, en caso de haber realizado solo cuatro movimientos (apertura de piezas negras, mueven las piezas blancas, mueven negras y mueven las piezas blancas de nuevo) el tiempo correspondiente (t) es igual a 3, independientemente del tiempo que le haya tomado a jugador decidir y llevar a cabo los movimientos.

Y finalmente, como hemos observado en el ejemplo anterior, un movimiento encadenado no aumenta el "t" de la partida, es decir, en caso de que el jugador de negras realice las capturas que se muestran en la figura 4.47, el "t" aumenta en 1, no en 3.

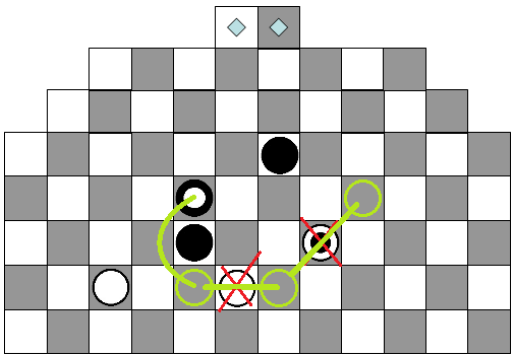


Figura 4. 47

Se escoge para el agente del tiempo el valor de i y no el de ply porque si utilizáramos el valor de este último, se proporcionaría una ventaja a las piezas negras que poseerían en cada jugadas (interviniendo ambos colores) un valor ply+1 respecto a las blancas, y dando más valor a algunos factores de su función de evaluación, ya que ***t*** actúa como un producto en la función de evaluación, de este modo, utilizando i, hay igualdad hacia los dos colores, el tiempo es el mismo, refiriéndose a la misma jugada.

A continuación se mostrara una breve descripción de cómo el tiempo influye en los factores anteriormente nombrados:

-Numero de piezas: está establecido como uno de los objetivo de juego eliminar las piezas del contrario hasta que solo posea una, ya que de este modo, la partida se da por finalizada por no poder conquistar las dos casillas de castillo. Un jugador se encuentra en ventaja directa si, tras haber realizado 4 movimientos, posee el doble de piezas que su oponente. A medida que transcurre la partida, y mayor es el número de i, es más importante mantener todas las piezas, te da una ventaja competitiva sobre el contrario. De este modo, con el índice añadido del tiempo, se da más valía al

factor, y BOU tiende a tomar decisiones en las que las que la protección de sus piezas es prioritaria.

-Proximidad al castillo: la proximidad al castillo de las piezas depende momento de la partida en que nos encontremos. Cuanto más avance el juego de manera natural las piezas deben ir tomando posiciones hacia la parte de tablero en la que se encuentra el castillo a conquistar, ya que, en caso de no poder capturar piezas enemigas, objetivo principal de BOU, sus movimientos, tienen que estar bien dirigidos. Al igual que el factor de la seguridad, con el avance de las piezas, a medida que se desarrolla el juego, con la aproximación al castillo estamos descartando la estrategia defensiva, útil para evitar el ataque del contrario, pero ineficiente para obtener la victoria. Con la relación de este factor con el tiempo, se iguala en importancia al número de piezas, que, junto con este, forman los factores que proporcionan la victoria a BOU.

Cómo se ha dicho antes, en el resto de factores también el tiempo podría influir de alguna manera, pero siempre de un modo menos significativo, por tanto el valor de ***t*** para ellos es igual a 1. Quedando la fórmula desglosada así:

$$F(e)=200(t_1).(f_1) + 75(f_2) + 50(f_3) + 200(t_4).((f_4) +50(f_5)+75(f_6) +175(f_7) +75f_8) + 75(f_9)+ 25(f_{10})$$

4.5.4. Modificaciones de la función de evaluación

La función de evaluación es una de las partes de BOU que más modificaciones ha sufrido a lo largo del proceso de creación:

Una vez se establecieron como definitivos los factores que componen la función principal (explicados más en detalle en el apartado 4.5.1 de éste capítulo), se procedió a escoger los pesos que los acompañaban, por no compartir una importancia heterogénea entre ellos. En este primer paso, no existían 10 factores, si no 9, el número de combos caballero-hombre era el factor número 4. La distribución inicial de estos pesos fue la siguiente:

Número de piezas (f_1):**1**

Seguridad (f_2):**3**

Número de caballeros (f_3):**2,5**

Número de combos caballero-hombre (f_4):**1,5**

Proximidad a los flancos (f_5):**0,75**

"Pelotones" de piezas (f_6):**1**

Castillo ya ocupado (f_7):**1,5**

"Concentración" de piezas (f_8):**1,5**

A continuación se procedió a aumentar el rango de los pesos para que cualquier variación decimal en los factores, resultara significativa en la elección de movimiento, por tanto, pasaron a encontrarse en un rango entre [0, 200], y sumar todos ellos 1000. Además se incluyó el factor de la movilidad, también importante en el desarrollo de cualquier partida.

Número de piezas (f_1):**100**

Seguridad (f_2):**200**

Número de caballeros (f_3):**100**

Número de combos caballero-hombre (f_4):**125**

Proximidad a los flancos (f_5):**50**

"Pelotones" de piezas (f_6):**75**

Castillo ya ocupado (f_7):**100**

"Concentración" de piezas (f_8):**100**

Movilidad (f_9):**150**

Una vez jugadas unas partidas con el juego de BOU, se llega a la conclusión de que se está dando más importancia a factores que no la tienen, ya que las piezas se mueven sin seguir un patrón lógico, tienden a realizar un movimiento arbitrario y a deshacerlo en el siguiente movimiento.

La ocupación de los castillos pasó de valer 100 a 200, porque si uno de los posibles movimientos lleva a ocupar el castillo, esta tiene que ser la elección escogida, hay que darle un valor añadido al movimiento. La seguridad pasa de poseer un peso de 200 a 100, ya que las piezas, en lugar de avanzar hacia el castillo, tendían a protegerse, y ese, aunque importante, no es un objetivo del juego. El número de caballeros pasa de tener un peso de 100 a 125, y el número de piezas totales, de 100 a 200, ya que se ha comprobado jugando la importancia del material sobre el tablero y el valor de este. Es

importante que las piezas muevan siempre tendiendo a proteger su material y eliminar el del contrario. El factor de número de combos caballero-hombre se elimina de la función ya que, se le estaba dando mucha valía a la velocidad de avance del caballero, gracias a la carga, pero las piezas de hombre, con el movimiento de *galope*, pueden avanzar a igual velocidad. De modo que los pesos quedarían así:

Número de piezas (f_1):**200**
Seguridad (f_2):**100**
Número de caballeros (f_3):**125**
Proximidad a los flancos (f_4):**50**
"Pelotones" de piezas (f_5):**75**
Castillo ya ocupado (f_6):**200**
"Concentración" de piezas (f_7):**100**
Movilidad (f_8):**150**

Por último, y siempre basados en la experiencia de juego, se llevaron a cabo una serie de modificaciones, tales como, el número de caballeros paso de valer 125 a 50, ya que no tenía tanta importancia el tipo de pieza que estuviéramos salvaguardando, tanto como el número de ellas. La concentración de los castillos pasó de valer 100 a 75, porque se comprobó que no importa cuánto de cerca se encuentren las piezas, es mas valioso que se encuentren en territorio enemigo; y la movilidad, paso de valer 150 a 75, porque no era algo tan influyente en el desarrollo de la partida que las piezas tuvieran más movilidad.

También se incluyó un nuevo factor, que se consideró de vital importancia para el juego; ya que, las piezas, aunque ya mejor dirigidas, seguían sin desplazarse claramente hacia el castillo enemigo, por tanto, con el factor de la proximidad, que cuenta el número de casillas hasta él, y dándole una importancia bastante alta, de 150, las piezas tratan de reducir ese número.

A continuación, se añadió el factor eliminado anteriormente de combos caballero-hombre, con un peso de 25, ya que, no influye demasiado en la toma de decisiones, pero con el valor de un caballero por encima de la pieza de hombre (explicado en el apartado 4.4.1. de este mismo capítulo) una combinación de una pieza

de hombre y caballero siempre permite hacer la carga del caballero, lo que puede resultar beneficioso. Quedaron entonces así los pesos de los factores:

Número de piezas (f_1):**200**
Seguridad (f_2):**100**
Número de caballeros (f_3):**50**
Proximidad al castillof_4):**150**
Proximidad a los flancos (f_5):**50**
"Pelotones" de piezas (f_6):**75**
Castillo ya ocupado (f_7):**200**
"Concentración" de piezas (f_8):**75**
Movilidad (f_9):**75**
Número de combos caballero-hombre (f_{10}):**25**

Por último, se llevo a cabo, una serie de modificaciones definitivas, debido a comportamientos erróneos en los desplazamientos de BOU.

El peso del factor de la proximidad al castillo pasa a valer 200, ya que, con el peso de 150 que poseía anteriormente, no se dirigían sus movimientos hacia el castillo enemigo en caso de no plantearse capturas de enemigos. Del mismo modo, la ocupación al castillo pasa a valer 175 en lugar de 200 y la seguridad pasa a valer 75, para equilibrar la suma de los pesos, debido a que el primero de estos factores no necesita un peso demasiado alto gracias a su dependencia con el tiempo de partida, y el segundo, la seguridad, provocaba conflictos en la toma de decisión de BOU, haciendo que tendiera a concentrarse en lugar de avanzar hacia el objetivo. De este modo quedan entonces así definitivamente los pesos de los factores:

Número de piezas (f_1):**200**
Seguridad (f_2):**75**
Número de caballeros (f_3):**50**
Proximidad al castillof_4):**200**
Proximidad a los flancos (f_5):**50**
"Pelotones" de piezas (f_6):**75**
Castillo ya ocupado (f_7):**175**

"Concentración" de piezas (f_8):**75**

Movilidad (f_9):**75**

Número de combos caballero-hombre (f_{10}):**25**

La función de evaluación es el agente más importante que toma parte en la inteligencia de los movimientos de BOU, por tanto, las repercusiones de modificarla influyen directamente sobre su funcionamiento. Se han establecido estos pesos como definitivos, pero la experiencia en el juego puede provocar la necesidad de cambios en ella, no existe una verdad escrita.

4.6. La búsqueda

Es de máxima importancia el buen funcionamiento y la buena estructuración del árbol de búsqueda para la toma de decisiones de BOU.

La búsqueda de un movimiento por parte de BOU ha evolucionado de la siguiente manera.

4.6.1. Fase uno. Movimiento al azar.

En esta fase para conseguir que BOU moviera una pieza llegado su turno en una partida, se ejecuta el método *Juega_BOU*, el cual, escogiendo una casilla al azar del tablero, se evalúa, si existe, se evalúa su contenido, y si posee una pieza del color con el que jugamos; esta operación se repite tantas veces como sea necesario hasta que encuentre una pieza de nuestro color.

Una vez conseguido esto, evalúa las casillas contiguas a la elegida y mueve la pieza al primer hueco disponible que encuentre, siguiendo este orden que se muestra en la figura 4.48.

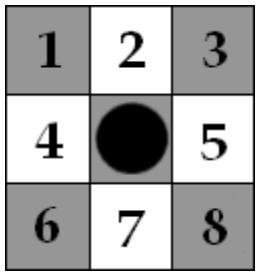


Figura 4. 48

Este método de búsqueda, es bastante ineficiente, ya que, BOU, no persigue ningún objetivo, conoce las reglas del juego y las cumple, pero, a no ser que sea fruto de la casualidad, no avanza hacia el castillo ni captura piezas enemigas.

4.6.2. Fase dos. Desplazamiento hacia el castillo

En esta fase se mejoran las condiciones de la búsqueda de movimiento de BOU con *Juega_BOU2*. En este caso, el método se encarga de recorrer el tablero desde la fila más próxima al castillo enemigo hasta la más alejada comprobando en cada casilla, hasta encontrar una pieza del color de BOU. De esta manera, al buscar ordenadamente las piezas, puede evitar la demora que puede suponer escoger casillas al azar, lo que, puede hacer que se repitan, o encontrar alguna pieza demasiado alejada del castillo objetivo. De este modo, la pieza que movamos, será la que esté más cerca del castillo oponente.

Una vez encontrada esta pieza, entre los 8 posibles movimientos que existen, se escoge uno de los tres movimientos que la desplacen hacia el castillo objetivo (como se indica en las piezas de la figura 4.49), de esta manera, nos aseguramos que nos estamos acercando y no alejando del castillo enemigo.

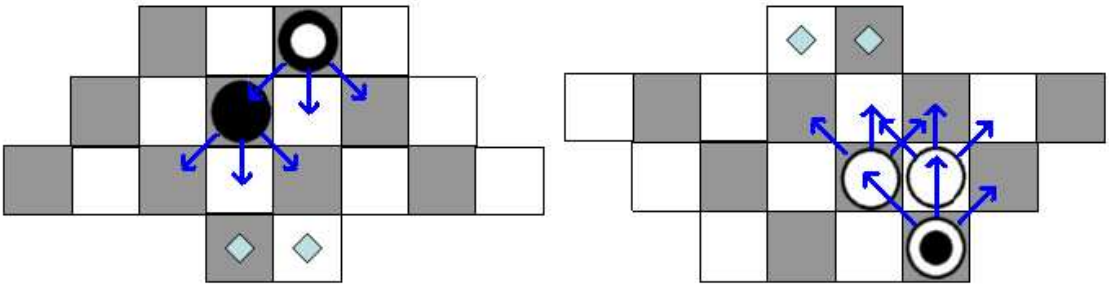


Figura 4. 49

En caso de no ser posible el desplazamiento hacia el castillo oponente, se hará hacia cualquier dirección.

A pesar de que este método muestra mejoras respecto al anterior, la eficiencia, sigue siendo insuficiente. Mientras que BOU solamente avanza hacia el castillo, el oponente puede ir capturando sus piezas, mientras que BOU solo lo haría si fuera fruto de la casualidad.

4.6.3. Fase tres. Búsqueda del movimiento óptimo

En busca de modificaciones para mejorar las decisiones de BOU, se crea el método de *recorre_arbol*, el cual, recorre el tablero buscando cada pieza de BOU y, a través de una función de evaluación, analiza la valía del tablero, para cada uno de sus posibles movimientos. Analiza uno a uno los movimientos, comparando su función de evaluación, y descartando aquellos que sean inferiores y manteniendo el movimiento con el mayor valor.

Con esta función, mejoramos notablemente el comportamiento de BOU, escoge, basándose en diferentes factores que afectan a la partida, el movimiento que más le conviene llevar a cabo.

4.6.4. Fase cuatro. Búsqueda teniendo en cuenta al oponente

Yendo un poco más lejos en la evaluación del los posibles movimientos con la función de evaluación, tenemos ahora en cuenta los movimientos que el oponente pueden realizar una vez hecha nuestra jugada; actuando a modo de predicción de sus movimientos.
Se evalúa, por cada movimiento que BOU puede hacer, un movimiento del oponente.

Este método no presenta demasiadas mejoras en el juego, ya que, escoge un movimiento al azar que el oponente podría realizar, en lugar de aquel movimiento que maximice su función de evaluación, ya que este, debería escoger por pura lógica, aquel movimiento que más le beneficie.

En vista de los resultados negativos obtenidos, se procede a programar algoritmos de búsqueda de mayor complejidad, tales como Minimax o Alfa-beta, ya explicados en los apartados 2.2.2.1. y 2.2.2.3.

4.7. Conclusiones

En este capítulo se ha recogido la parte de análisis y diseño del programa asi como las características a destacar del desarrollo del código la parte más costosa y que más tiempo ha consumido de la creación de BOU.

El diagrama de casos de uso y su descripción fue la primera parte que se llevó a cabo, pero como el sistema ha sufrido modificaciones, por tanto ha habido que generar varias versiones, sobretodo del esquema de los casos de uso

La primera versión del diagrama de clases, debido a las necesidades que surgieron con el programa, no fue más que un borrador del diagrama final que ha resultado. Sobretodo al finalizar el interfaz gráfico del juego, con el cual se incluyeron 7 clases nuevas. Todas las clases que lo conformas tienen importancia pero sobretodo lo tienen las clases de tableroGUI y casillasGUI , gracias a las cuales el usuario puedo interactuar con el tablero, y las encargadas de los movimientos y la generación de la respuesta de BOU y Panel4, que forma parte de la interfaz, y lleva todo el peso de la partida.

La parte programada fue la más complicada pero la más emocionante ya que se comienzan a verse resultados de los diseños realizados al comienzo del proyecto. La introducción de la función de evaluación con la elección de los factores que la componían y a su vez lo pesos que influían sobre ellos ha sido una de las fases también más elaborada y que ha llevado más tiempo, por tener que combinar la modificación del código con las pruebas de juego hasta dar con la combinación que producía una buena respuesta de BOU. AL final después de muchas modificaciones, los factores que más importancia tienen en BOU son *número de piezas, ocupación de catillos* y *proximidad al castillo*, factores que incitan a BOU a comer piezas enemigas, avanzar hacia el castillo enemigo en sus desplazamientos y a ocupar el castillo cuando se encuentre en sus inmediaciones. Incluir el número de jugada a la función de evaluación ha sido un importante avance, ya que a medida que transcurría el juego, se multiplica el valor de estos factores, que coincides con los más influyentes, los antes mencionados.

En cuanto al algoritmo de búsqueda, como se ha explicado al final de este capítulo ha evolucionado mucho desde el comienzo y las pruebas han demostrado, cual es el algoritmo que le proporción más valía al juego, Alfa-beta, y la profundidad que lo vuelve competitivo compensando con un buen tiempo de respuesta, 3.

Vemos en el siguiente capítulo las pruebas de juego aplicando los diferentes algoritmos y características explicadas en este.

Capítulo 5. Resultados

Para comprobar el buen funcionamiento de BOU ante cualquier situación, se han llevado a cabo una serie de pruebas y experimentos que analizan su comportamiento y lo evalúan. Este capítulo recoge las pruebas a las que ha sido sometido el programa desde las primeras a nivel más específico y sencillo, hasta alcanzar el mayor nivel de complicación.

5.1. La función *"escoger_capturas"*

Una de las reglas establecidas como oficiales del juego de Camelot, es que una pieza en el tablero está obligada a comer, siempre que le sea posible, antes de realizar cualquier otro movimiento. También está establecido que un jugador no tiene porque llevar a cabo un salto, aunque se dé cuenta de ello, si su contrincante no se ha percatado de la jugada.

Al reducir de algún modo el número de descendientes los algoritmos proporcionan una respuesta más rápida, y además, escogen aquel movimiento que le produce más valía a nuestra partida, a la vez asegurándonos de que se trata de una captura. El método de "escoger_capturas" se encarga de la selección de las capturas en los algoritmos de búsqueda utilizados en este programa.

Pero, debido a la reducción de la lista de movimientos a expandir, reducimos por tanto también, el número de nodos y tiempo en el algoritmo Minimax; llegando a encontrarnos resultados tales como los que se exponen a continuación, que desencajan totalmente con la dinámica del algoritmo, no podemos hablar de datos heterogéneo, existen picos:

Minimax	**Tiempo (sg)**	**Nº nodos expandidos**
	798, 63	18139
	1998,63	35588
	13,7	**233**
	2596,92	45321

Tabla 5.1

Este fenómeno afectará sobre los resultados que se presentan a continuación, en los siguientes apartados.

5.2. Fase de pruebas inicial

En lugar de colocar las piezas según la disposición oficial de inicio de partida, en estas pruebas se ubica a las piezas en situaciones más concretas, en las que, toman parte

menor número de ellas y están en posiciones ya próximas al castillo objetivo, provocando también la combinación de movimientos combinados, encadenados y más complicados, cosa que no se consigue hasta bien avanzado el juego.

Estas pruebas se han llevado a cabo con los dos algoritmos principales, Minimax y Alfa-beta

5.2.1. A profundidad 1

Esta es la fase inicial de las pruebas, se ha tomado cuatro situaciones que van complicando poco a poco el listado de posibles movimientos, para observar cómo reacciona BOU ante las diferentes posibilidades.

5.2.1.1. Movimiento de captura

Se muestra en la figura 5.1 una jugada en la que toman parte dos piezas atacantes y dos defensoras del castillo y BOU juega con las piezas negras, por tanto, el listado de posibles movimientos es el que se muestra en la parte derecha de la misma.

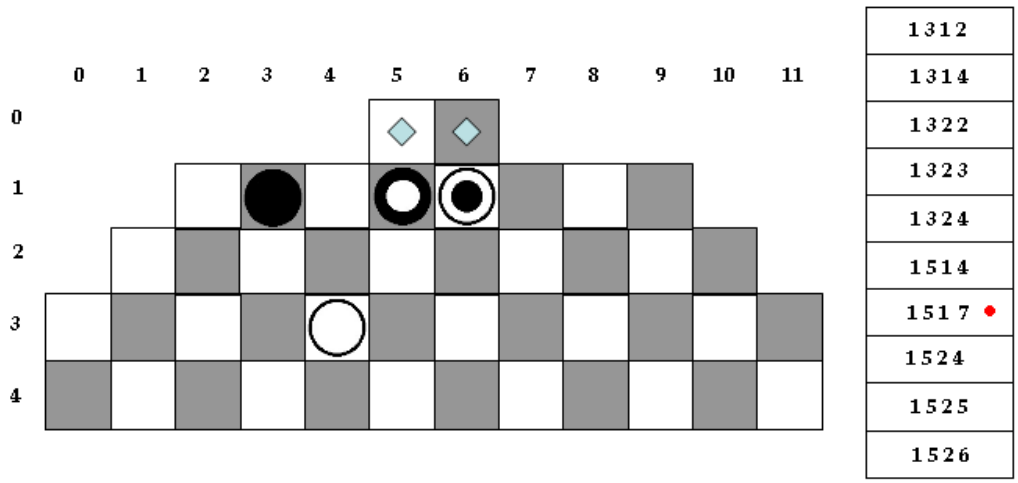


Figura 5.1

El resultado:

```
Turno de BOU
BOU: 1. 1,5 --> 1,7
Se han evaluado 1 nodos
Tiempo transcurrido: 0.0037451690000000004 sg
Fin del juego. Negras ganan.
```

BOU escoge el movimiento de captura: **1 5 1 7**, por ser, de todos los que hay, el que proporciona más valor al tablero resultante. Como se puede observar, el tiempo es

inapreciable, y solo se expande un nodo, por ser la única captura que existe en la lista, según el fenómeno de "*escoger_capturas*" explicado en el apartado 5.1

5.2.1.2. Movimiento de captura y carga del caballero

En este caso, se muestra en la figura 5.2 de la parte superior una disposición de las piezas que da lugar a un par de capturas y un movimiento encadenado. BOU, juega con las piezas negras y en la parte derecha se muestra un despliegue de los posibles movimientos que se pueden llevar a cabo.

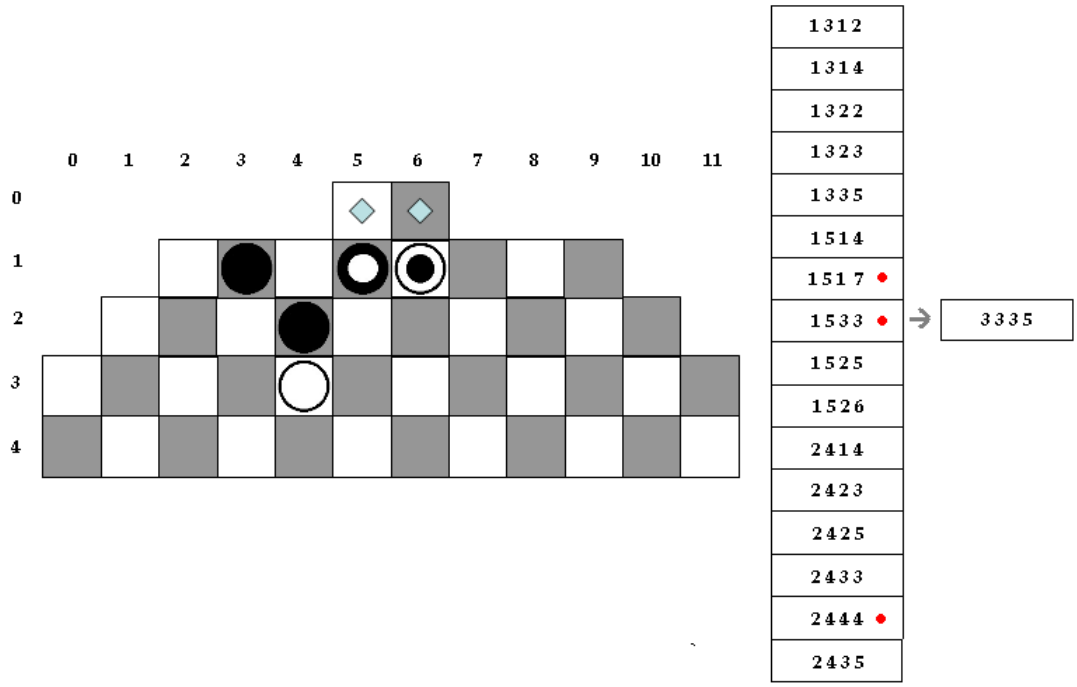


Figura 5.2

El resultado del programa es el siguiente:

Turno de BOU

BOU: 1. 1,5 --> 3,3 --> 3,5

Se han evaluado 1 nodos

Tiempo transcurrido: 0.007329703000000001 sg

En este caso, BOU, escoge el movimiento encadenado de **1 5 3 3 → 3 3 3 5**, porque proporciona más valía al tablero resultante, ya que, además de capturar una pieza enemiga, disminuye la proximidad al castillo enemigo (justo al otro lado del tablero), y es otro de los factores más influyentes en la función de evaluación.

5.2.1.3. Movimiento de captura, carga del caballero y movimientos encadenados

En este caso (figura 5.3) hay más variedad de movimientos y se han complicado los desplazamientos encadenados, así como también han aumentado el número de capturas. El listado que se presenta bajo el tablero, muestra el resultado final de la lista de posibles movimientos.

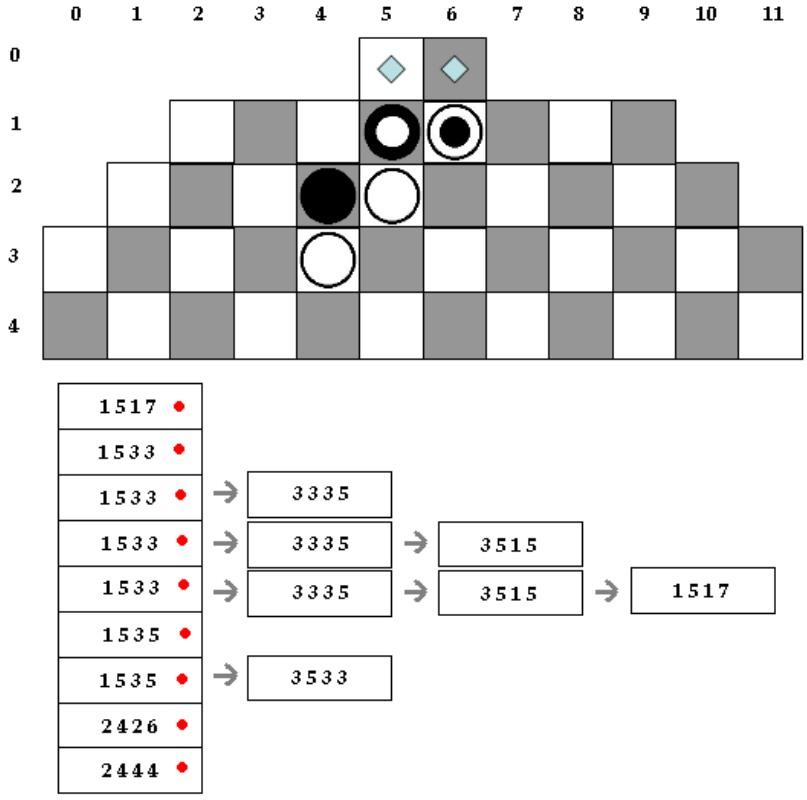


Figura 5.3

El resultado del programa es:

Turno de BOU

BOU: 1. 1,5 --> 3,3 --> 3,5 --> 1,5 --> 1,7

Se han evaluado 1 nodos

Tiempo transcurrido: 0.034638481 sg

Finalmente, evaluando los tableros resultantes tras realizar cada movimiento, aquel que proporciona más valor es **1 5 3 3 → 3 3 3 5 → 3 5 1 5 → 1 5 1 7,** porque existen otros movimientos que aproximan las piezas al tablero, pero el factor del número de piezas siempre aporta más valor, y más tratándose de un movimiento encadenado con 3 capturas.

5.2.2. A profundidad 3

Esta es la segunda fase de pruebas, en las que, a profundidad 3, se comprueba cómo reacciona BOU ante diferentes situaciones en las que se le plantea escoger entre unos movimientos u otros, proporcionando diferentes valores de función de evaluación, y teniendo en cuenta la posible respuesta de su oponente.

En este caso, tomaremos solamente dos pruebas pero con un poco más de complejidad que las realizadas a profundidad 1, ya que en el primer movimiento, casi todas las piezas en juego desaparecen del tablero, imposibilitando el evaluar el siguiente movimiento del oponente.

5.2.2.1. Captura encadenada

Presentamos en la figura 5.4 un caso de captura encadenada

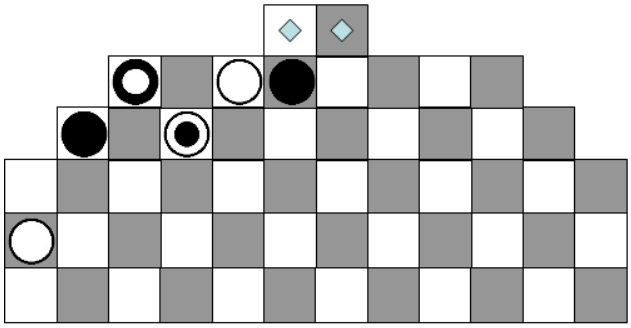


Figura 5.4

La respuesta de BOU ha sido:

Turno de BOU
BOU: 1. 1,5 --> 1,3 --> 3,3
Se han evaluado 1 nodos
Tiempo transcurrido: 0.010016549000000001 sg

En este caso, el algoritmo utilizado para el ejemplo ha sido Minimax. Pero también se ha comprobado sobre los mismo datos del tablero, la decisión sobre el algoritmo Alfa-beta. Y la respuesta ha sido:

Turno de BOU
BOU: 1. 1,5 --> 1,3 --> 3,3
Se han evaluado 1 nodos
Tiempo transcurrido: 0.026478601 sg

A profundidad 3 ya deberían notarse ciertas diferencias respecto a Alfa-beta, algoritmo más eficiente, pero dado que el ejemplo no presenta muchas piezas en juego, y la resolución de la partida es casi inmediato, el número de nodos expandidos es similar para ambos algoritmos, la poda que hace a Alfa-beta un algoritmo ventajoso, no tiene oportunidad de presentarse.

5.2.2.2. Captura encadenada 2

En este caso la figura 5.5 muestra una captura encadenada con mas piezas en juego.

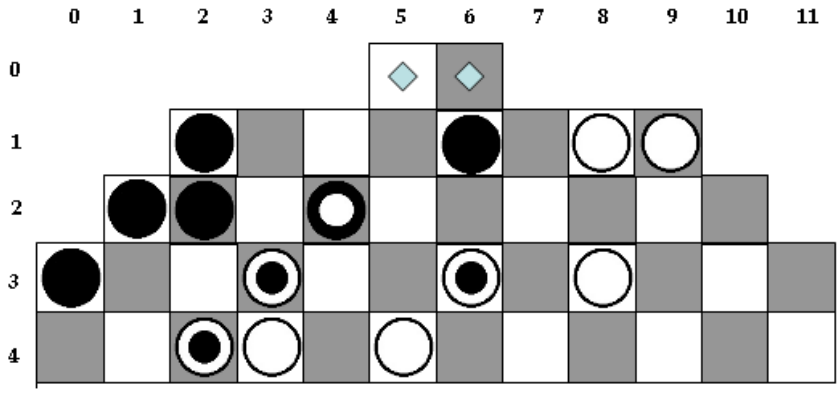


Figura 5.5

Presentando este tablero ejemplo en el que participan mayor número de piezas, y por tanto, hay mayor número de posibilidades de movimiento, BOU con el algoritmo Minimax, y con las piezas negras, ha respondido de la siguiente manera

Turno de BOU
BOU: 1. 2,2 --> 4,4 --> 4,6 --> 2,6
Se han evaluado 40 nodos
Tiempo transcurrido: 0.323009855 sg

Y del mismo modo, con la misma disposición de piezas, pero con el algoritmo Alfa-beta, la respuesta ha sido:

Turno de BOU
BOU: 1. 2,2 --> 4,4 --> 4,6 --> 2,6
Se han evaluado 40 nodos
Tiempo transcurrido: 0.267157876 sg

Llegamos a la conclusión, tras observar estos dos ejemplos, que las diferencias entre ambos algoritmos, con un pequeño número de piezas en juego, y en caso de posible captura, son ínfimas.

5.2.3. A profundidad 4

Finalmente, tomamos profundidad 4 como último y más alto valor, ya que, el tiempo de respuesta a esta profundidad, dado el número de nodos expandidos que se generan, aumenta notablemente. Por tanto, pasamos a comprobar dos situaciones, una en las que toman parte un número pequeño de piezas, para simplificar los resultados y

otro ejemplo en el que no haya captura inmediata, para que se evalúen todos los hijos descendientes.

5.2.3.1. Captura encadenada

Tomamos el segundo tablero ejemplo del apartado anterior (Figura 5.6) y probamos la respuesta de bou para el algoritmo Minimax a profundidad 4.

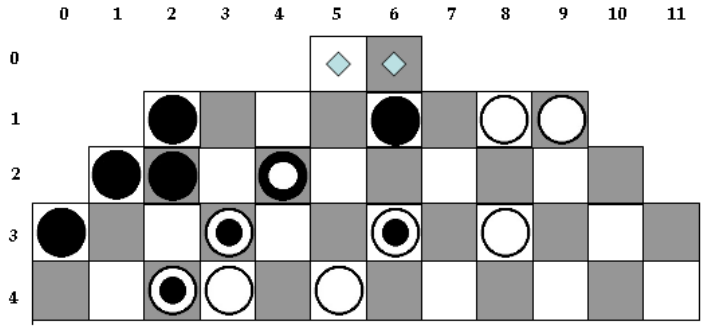


Figura 5.6

Turno de BOU
BOU: 1. 2,2 --> 4,4 --> 4,6
Se han evaluado 809 nodos
Tiempo transcurrido: 3.8541173300000002 sg

En cambio, el mismo ejemplo a profundidad 4, para el algoritmo Alfa-beta, obtiene la siguiente respuesta:

Turno de BOU
BOU: 1. 2,2 --> 4,4 --> 4,6
Se han evaluado 38 nodos
Tiempo transcurrido: 0.27722677700000004 sg

Se puede observar, ya a profundidad 4, como Alfa-beta, obtiene el mismo resultado que el anterior algoritmo, pero expandiendo menos nodos y en menor tiempo.

5.2.3.2. Movimiento sin captura

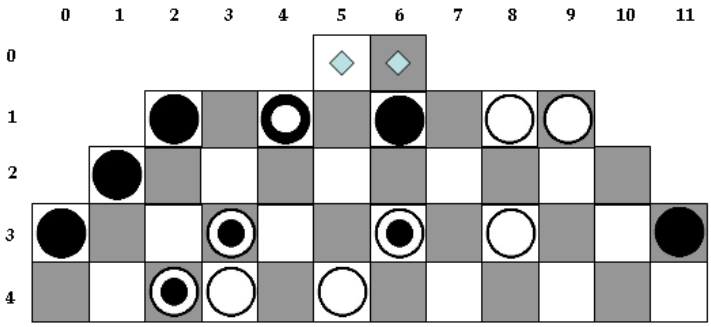


Figura 5.7

Veamos en este ejemplo (figura 5.7) la respuesta de BOU, moviendo piezas negras y con el algoritmo Minimax, en la que no hay ninguna captura inmediata:

```
Turno de BOU
BOU: 1. 2,1 --> 3,2
Se han evaluado 1633 nodos
Tiempo transcurrido: 11.812209560000001 sg
```

Y el mismo aplicado con el algoritmo alfa-Beta:

```
Turno de BOU
BOU: 1. 2,1 --> 3,2
Se han evaluado 549 nodos
Tiempo transcurrido: 2.6523389500000003 sg
```

Finalmente, se puede observarla igual, pero más rápida respuesta del algoritmo alfa-beta

5.3. Usuario vs BOU

En este apartado se tratarán dos conceptos en función de los cuales se presentarán comparaciones entre los algoritmos utilizados. Uno de ellos es el **tiempo de respuesta,** tiempo que tarda BOU en generar los descendientes del nodo origen y expandir todos sus hijos correspondientes, evaluando su valor y escogiendo aquel que le proporciona un mejor valor; y el número de **nodos expandidos,** como su nombre indica, es el número de nodos que se expanden en el recorrido del árbol de búsqueda.

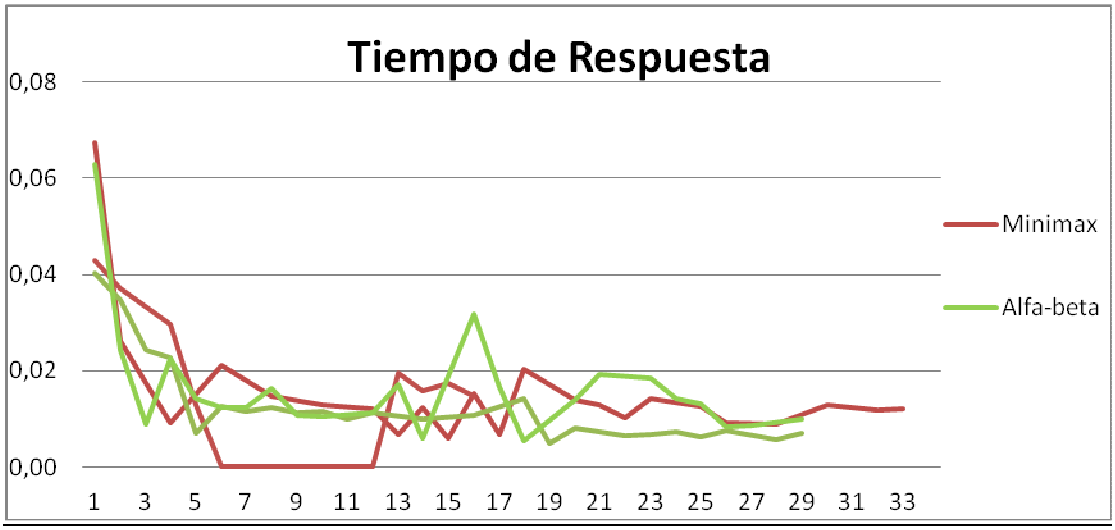
Dependiendo de la gráfica el eje vertical corresponderá con uno de estos dos conceptos, pero el eje horizontal representa cada uno de los turnos de BOU en una partida contra un usuario.

5.3.1. Cambios de profundidad

En este punto se procederá a hacer una serie de comparaciones en función de la profundidad del árbol de búsqueda.

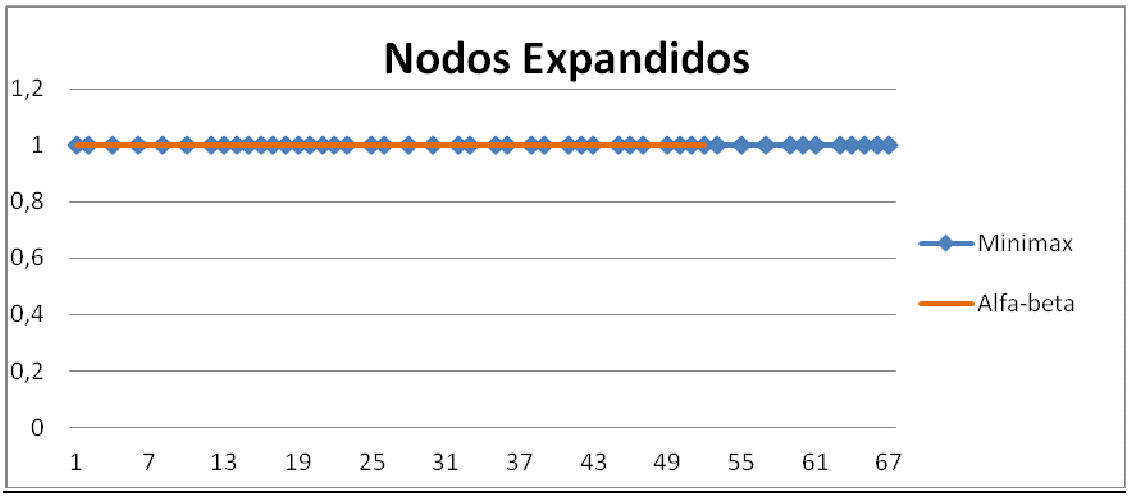
5.3.1.1. Profundidad 1

A continuación se representa en dos graficas la respuesta de BOU en pleno juego en cuyo árbol de búsqueda se ha establecido a profundidad 1. En este nivel el factor de ramificación es de 112, 14 piezas a evaluar con 8 posibles movimientos cada uno. La primera de las gráficas (gráfica 6.1), muestra varios ejemplos de tiempo de respuesta de BOU a profundidad 1, se representan dos casos de partida con el algoritmo Minimax, y dos casos con el algoritmo Alfa-beta en juego.



Gráfica 6. 1

A esta profundidad las diferencias de tiempo entre ambos algoritmos son inapreciables, por tanto se puede observar en la gráfica como ambos algoritmos tienen comportamientos similares. Ambos comienzan con un tiempo mayor de respuesta porque al no existir la posibilidad de capturas, BOU expande todos los nodos de su árbol comparando el valor de cada uno de ellos. Además, presenta puntos en los que el tiempo desciende a valores muy próximos a 0 porque el número de descendientes se reduce a un número muy limitado debido al fenómeno explicado apartado 5.1, con el cual, se escogen las capturas o los movimientos que llevan a final del partida por encima del resto.



Gráfica 6. 2

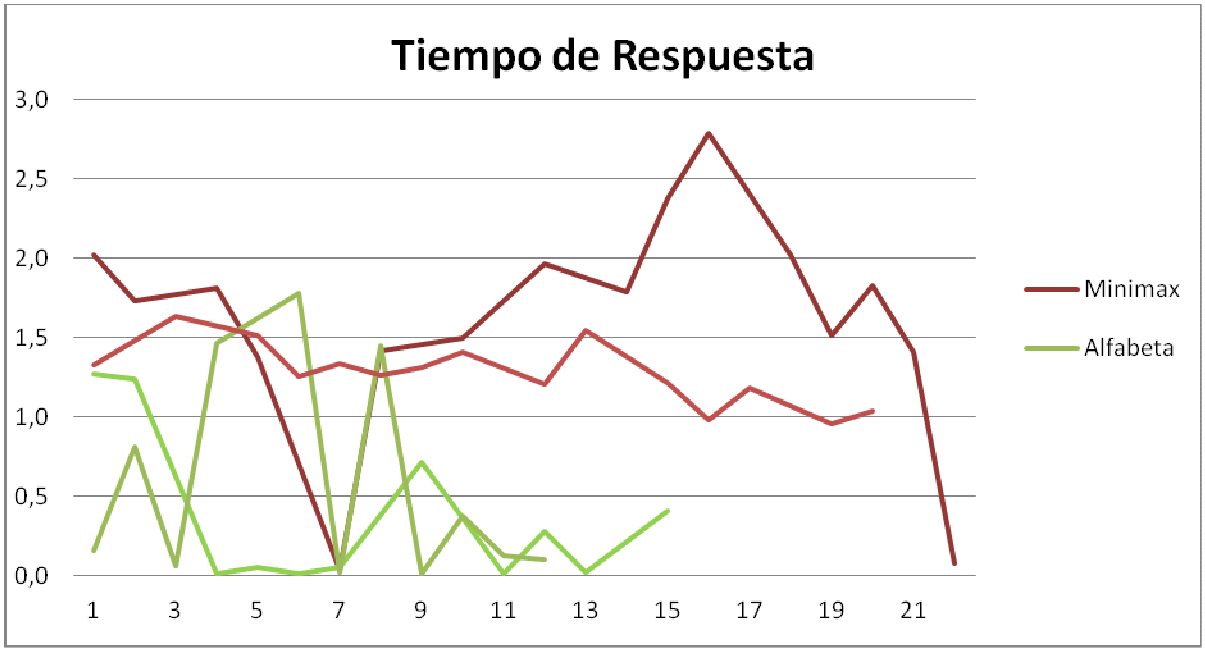
En cuanto al número de nodos expandidos, a profundidad 1, independientemente del algoritmo que se utilice, siempre será de uno, ya que solamente

se hallan los descendientes del tablero en el que nos encontramos, no se tienen en cuenta los siguientes pasos. De este modo, los cuatro casos representados en la grafica 6.2 (dos partidas jugadas con el algoritmo Minimax y dos con el algoritmo Alfa-beta),sitúan para cada uno de los turnos el número de nodos en la línea del valor 1.

5.3.1.2. Profundidad 2

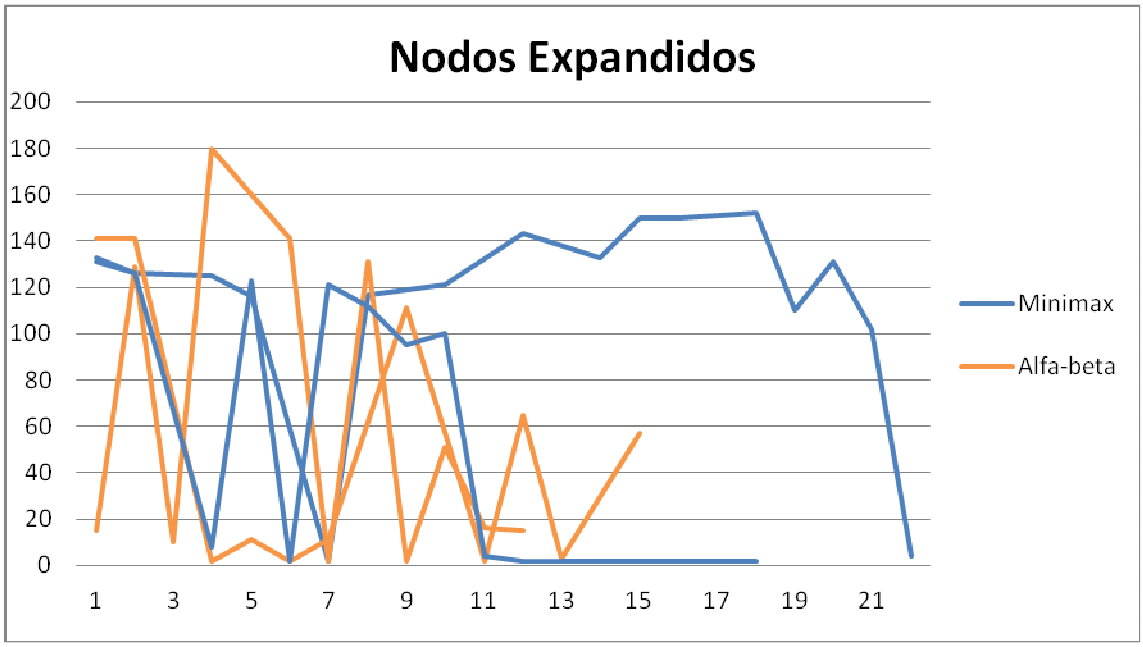
A continuación se muestra las dos gráficas correspondientes a los conceptos de tiempo de respuesta y nodos expandidos a una profundidad de 2 en partidas aleatorias de los dos algoritmos que estamos comparando. En este nivel el factor de ramificación es de 12.544 nodos, 112 posibles movimientos, para cada posible movimiento de los 112 movimientos del nivel anterior (112^2).

Como se puede observar en la grafica 6.3, en cuando a la rapidez de respuesta por parte de ambos algoritmos, en los dos ejemplos representados ya se muestras ciertas diferencias entre ambos.



Gráfica 6. 3

Los casos representados con el color rojo demuestran que Minimax, respuesta con mayor lentitud que Alfa-beta, cuyos ejemplos están representados con el color verde. La diferencia no es demasiado amplia, porque se trata de tiempos de respuesta inferiores a 3 segundos, por tanto, en realidad, en pleno juego, no se nota mejoría de un algoritmo respecto a otro.



Gráfica 6. 4

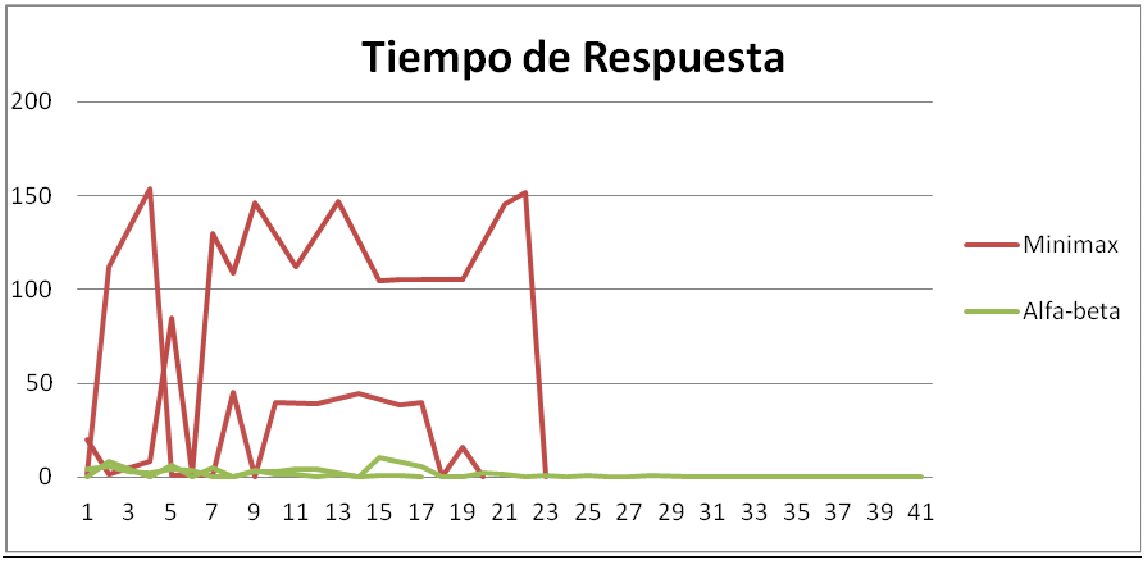
Se observa ahora la comparación en cuanto al número de nodos expandidos por ambos algoritmos en la gráfica 6.4. De nuevo, se puede apreciar que esta gráfica es atípica, se vuelve a repetir los casos de turnos en los que solo se expanden 2, 10, o en cualquier caso un número de nodos inferior a 20, debido a las posiciones con posibilidad de captura, que descarta el resto de movimientos.

De todos modos, a pesar de este fenómeno, que provoca tantos picos en la gráfica 6.4, se observa un comportamiento, medianamente normal en el que, en la mayoría de los casos, el comportamiento de los dos ejemplos con el algoritmo Alfa-beta se encuentran por debajo del número de nodos expandido por el algoritmo Minimax.

5.3.1.3. Profundidad 3

Presentamos a continuación las gráficas correspondientes a las cuatro partidas jugadas, dos con el algoritmo Minimax, y dos con el algoritmo Alfa-beta a profundidad 3. El factor de ramificación a este nivel es de 1.404.928, obtenidos de los 112^3.

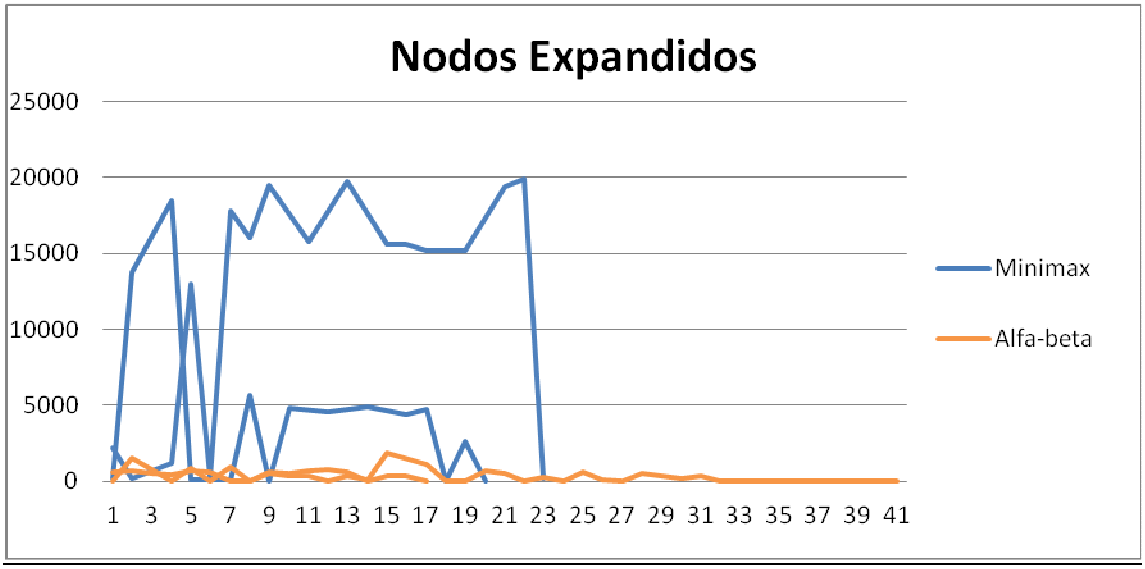
Como se observa en la grafica 6.5, a esta profundidad del árbol de búsqueda, sí que se presentan cambios significativos entre algoritmos, Minimax, llega a alcanzar los dos minutos y media de tiempo de respuesta en uno de los ejemplos, mientras que los dos ejemplos del algoritmo Alfa-beta apenas alcanzan los 20 segundos de respuesta.



Gráfica 6. 5

Y del mismo modo, en la grafica 6.6 se muestra de nuevo una notable diferencia entre el número de nodos expandidos por el algoritmo Minimax respecto de los expandidos por el algoritmo Alfa-beta, llegando el primero a alcanzar los 20000 nodos generados en uno de los ejemplos, mientras que el algoritmo Alfa-beta apenas sobrepasa los 2000 nodos expandidos.

Siguen observándose importantes picos en el transcurso de la partida para ambos algoritmos, con motivo del descarte de movimientos, siendo más significativos para el algoritmo Minimax, y que sus búsquedas oscilan entre números muy altos y muy bajos del orden de 15000 nodos de diferencia.



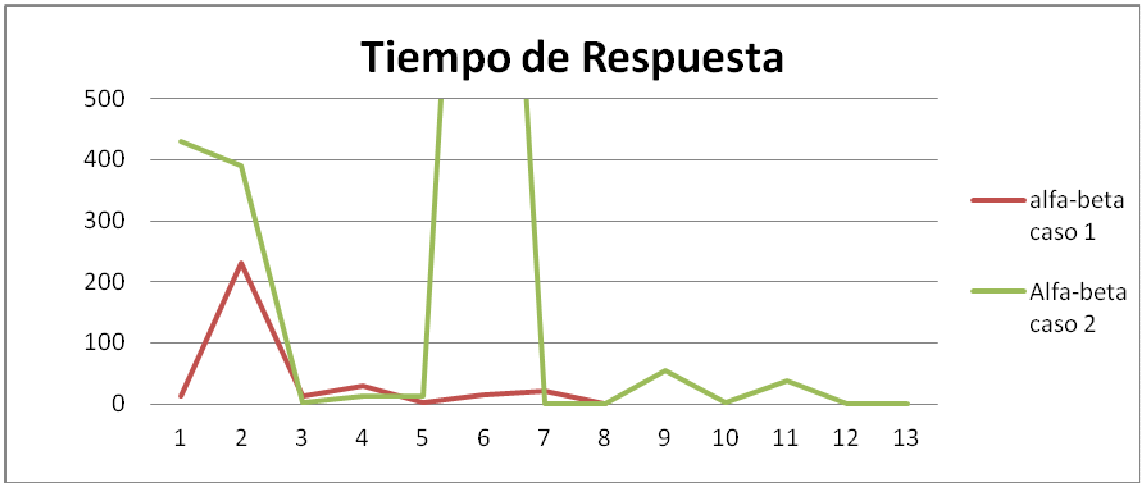
Gráfica 6. 6

5.3.1.4. Profundidad 4

Por último, se muestra un ejemplo de comportamiento de BOU estableciendo la profundidad del árbol de búsqueda a 4 (factor de ramificación 112^4 es igual a 157.351.936).

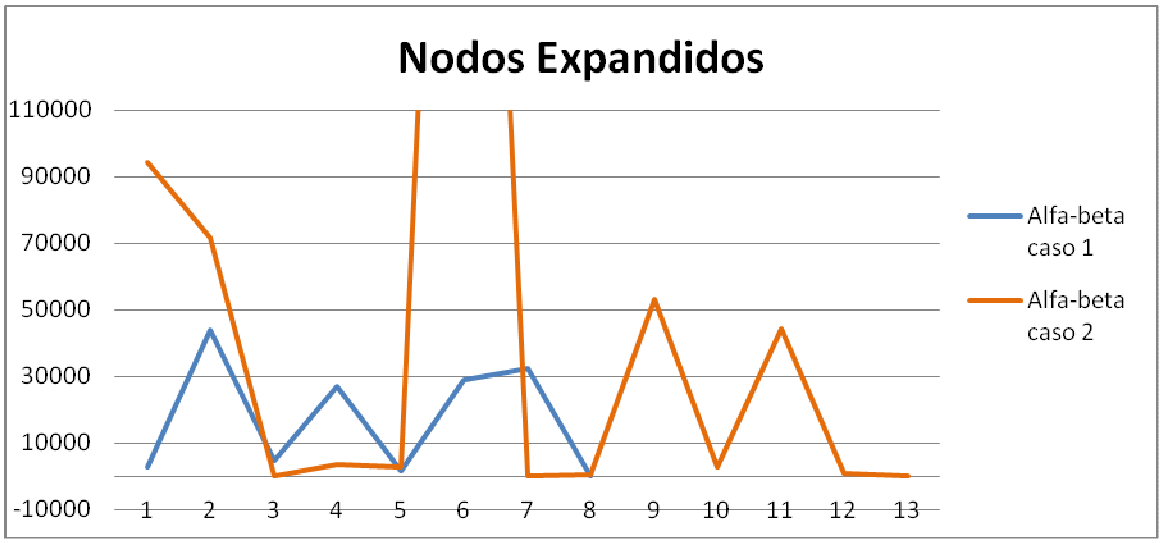
Tratando de obtener resultados de juego con el algoritmo Minimax a profundidad 4 nos encontramos con el siguiente obstáculo: el tiempo de respuesta es superior a 43.200 segundos, es decir, para obtener un movimiento por parte de BOU el usuario debe esperar al menos 12 horas. Lo que significa que, en una partida media, para cuya resolución se llevaran a cabo 24 movimientos, teniendo en cuenta que el usuario resuelve sus jugadas en un minuto, la partida llevaría 8.652 minutos, es decir, 144,2 horas, o lo que es lo mismo, algo más de 6 días para acabar la partida. Por tanto, se descarta la obtención de resultados a profundidad 4 para este algoritmo.

Se han llevado a cabo numerosas pruebas para obtener resultados de partidas con el algoritmo Alfa-beta a profundidad 4 y se han presentado inconvenientes debido al alto consumo de memoria del programa, por el alto factor de ramificación. La gráfica 6.7 muestra el comportamiento del algoritmo alfa-beta en dos partidas a profundidad 4. Presenta valores mucho menores que el algoritmo Minimax, pero a su vez, bastante mayores que los resultados a profundidad 3 de la gráfica 6.5. El caso número 1 presenta valores más pequeños de tiempo debido a que han sido respuesta a movimientos de captura. En cambio el caso número dos presenta valores mucho mayores, por ser el resultado de que el usuario haya evitado las piezas enemigas, obligando así a BOU a expandir todos sus nodos.



Gráfica 6. 7

En la jugada número 6 el valor de tiempo excede el límite del eje, establecido así para poder apreciar los valores más pequeños de los dos casos. El tiempo en ese turno alcanza la cota de 1662,32 segundos, es decir, 27 minutos de espera por la respuesta de BOU. Siguen presentando oscilaciones, en las que los valores de tiempo se aproximan a 0 debido a la selección de las capturas.



Gráfica 6. 8

Finalmente, en la Gráfica 6.8 se representa el número de nodos en una partida completa, observando que se alcanzan ya valores del orden de 100.00, por la cantidad de nodos que se deben evaluar a esta profundidad, y de la misma manera que se ha producido en el tiempo de respuesta, en el turno 6, el número de nodos expandidos alcanza el valor de 397.241, valor que se ha dejado fuera del gráfico para poder apreciar el más detalle aquellos valores más pequeños del resto de la partida.

Algoritmo	**Color** (BOU)	**Profundidad**	**Media tiempo(sg)**	**Media nº nodos**
Minimax	Negras	1	0,0198	1
	Blancas	1	0,0142	1
	Negras	2	0,5799	62,909
	Blancas	2	0,372	49,636
	Negras	3	33,5973	4179,889
	Blancas	3	68,1215	9095,764

Algoritmo	Color (BOU)	Profundidad	Media tiempo(sg)	Media nº nodos
Alphabeta	Negras	1	0,01397	1
	Blancas	1	0,0178	1
	Negras	2	1,1942	55,155
	Blancas	2	0,862	82,433
	Negras	3	3,0433	568,294
	Blancas	3	1,6484	388,098
	Negras	4	40,4957	17651,625
	Blancas	4	200,2514	51626,154

Tabla 5.2

En resumen, la tabla 5.2, recoge las medias de los valores que se han recogido para realizar las gráficas de los apartados anteriormente explicados.

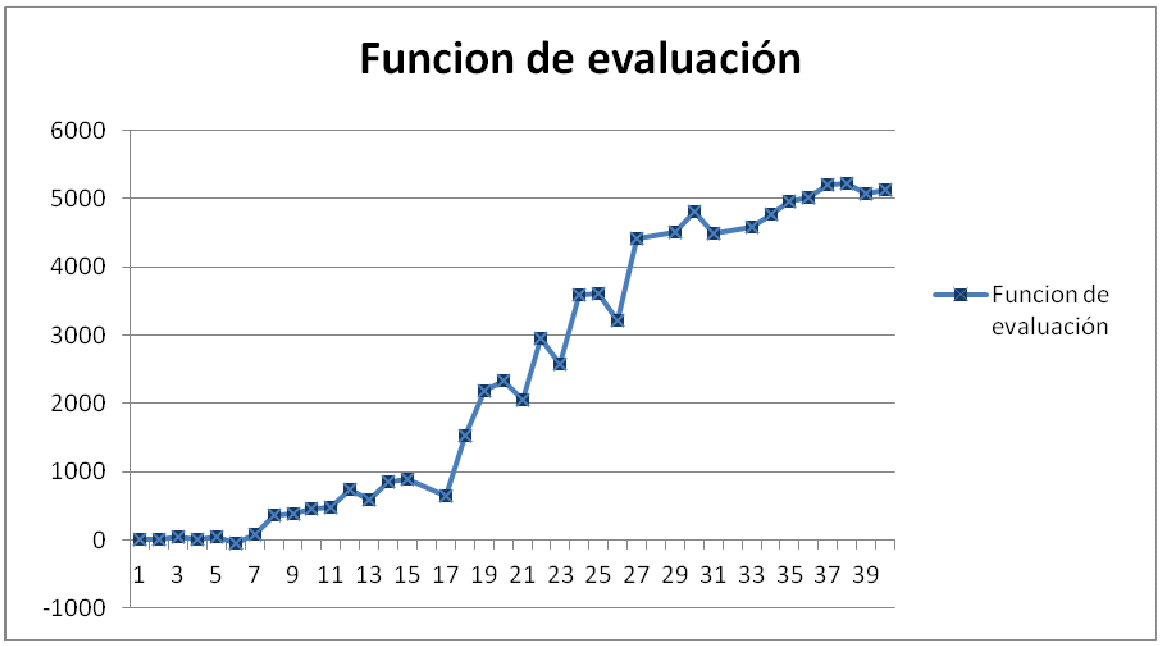
En los ejemplos en los que BOU juega con las piezas blancas, y por tanto realiza la apertura de juego, siempre lo hace de la misma manera, escoge el mismo movimiento, ya que la disposición es siempre igual, y el movimiento óptimo siempre será el mismo, por eso el usuario, ha tomado diferentes elecciones, para obtener también diferentes respuestas por parte de BOU

En la tabla se puede observar que al hacer las medias del tiempo de respuesta y el número de nodos expandidos, se ha diferenciado (además de por algoritmos y por profundidad), según el color de piezas que BOU haya capitaneado en la partida. El motivo de esta distinción no es otro que el de encontrar algún tipo de comportamiento anómalo o diferente en función de la prioridad de movimiento o el comienzo de partida. Como se puede observar, los resultados obtenidos indican que las diferencias son sutiles respecto a este criterio, y probablemente sean más debidas a los diferentes movimientos realizados por el usuario, o el propio azar de los acontecimientos, porque finalmente, las más notables diferencias son las relacionadas con la profundidad del árbol de búsqueda.

5.3.2. La función de evaluación

En la gráfica a continuación se muestra la evolución de la función de evaluación en una partida de 40 jugadas. Se ha escogido una partida tipo en la que BOU obtiene la victoria, a profundidad 3, y con el algoritmo Alfa-beta; se han estudiado posibles diferencias a distinta profundidad y con el algoritmo Minimax, pero no se presenta nada significativo. Por tanto, la evolución tipo es la que se muestra a continuación en la gráfica 6.9.

Se puede observar la evolución positiva de la partida para BOU ya que el valor de la función es ascendente; cuando ésta toma valores negativos (al comienzo de la partida), así como cuando se producen incrementos del valor, muestra que el usuario ha tomado cierta ventaja y recupera valor en la partida, pero finalmente, termina por decantarse al triunfo de la maquina.



Gráfica 6. 9

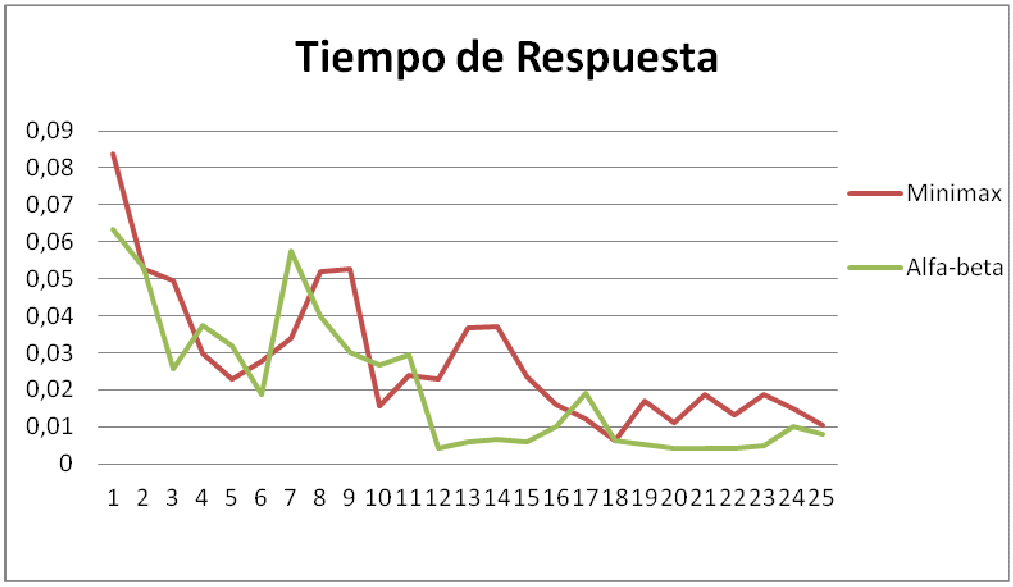
5.4. BOU vs BOU

En este apartado enfrentamos a los dos algoritmos principales en una partida, Minimax y Alfa-beta. Se realizarán pruebas a tres profundidades diferentes y solamente una partida ya que, partiendo de la misma disposición de las piezas, las decisiones tomadas serán iguales siempre, para ambos. Se prueba esta modalidad debido a que , en las partida de BOU contra el usuario, este puede presentar un variable nivel de juego.

5.4.1. Profundidad 1

A continuación se procede a generar las gráficas a partir de los datos obtenidos de enfrentar en una partida al algoritmo Minimax y Alfa-beta, a profundidad 1.

La gráfica 6.10 muestra una partida de 25 movimientos que finalizó con la victoria del algoritmo Alfa-beta. Cabe destacar el comportamiento lineal descendente, debido a la captura sucesiva de las piezas, a medida que existen menos piezas en juego disminuye el tiempo de respuesta.



Gráfica 6. 10

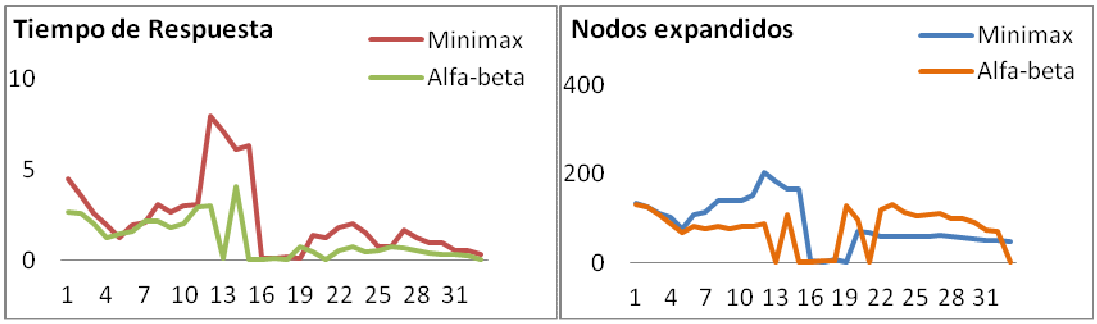
Se ha descartado incluir la gráfica del número de nodos expandidos ya que, este permanece con el valor de 1 para los dos algoritmos, y durante todo el transcurso de la partida, por tanto, no es representativa.

5.4.2. Profundidad 2

A continuación se verá una muestra de una partida jugada entre ambos algoritmos a profundidad 2.

La grafica 6.11 plasma los resultados de una partida de 32 jugadas que se soluciona con la victoria de Alfa-beta. Estudiando los dos conceptos conjuntamente se puede apreciar como los comportamientos de ambos son similares, en altos valores de tiempo, mayor número de nodos expandidos.

No se aprecian grandes diferencias, pero alfa-beta, tiene mejor respuesta que Minimax en las gráficas 6.11, sus valores, salvo en ciertas excepciones, se mantienen por debajo de los valores de Minimax.



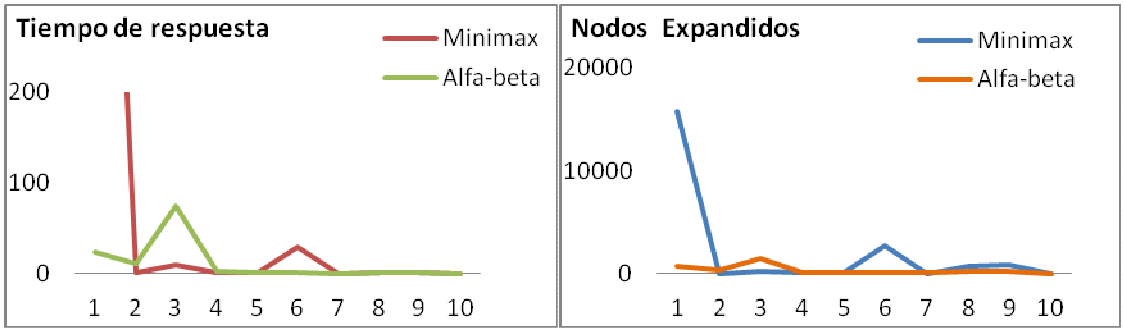
Gráfica 6. 11

5.4.3. Profundidad 3

Por último se estudiará el comportamiento de los dos algoritmos en una partida a profundidad 3.

La gráfica 6.12 muestra una partida de 10 jugadas cuya victoria pertenece a Alfa-beta.

las gráficas de los conceptos que se analizan tiene comportamientos análogos, en los distintos momentos de juego. Cabe destacar el comienzo de partida en el que el algoritmo Minimax, ha alcanzado el valor de 15811 nodos expandidos y casi 889 segundos de respuesta debido a que, en esta situación de apertura de movimiento, en la que no hay posibilidad de captura y todas las piezas se encuentran en juego, nos encontramos en el peor de los casos del análisis. En los siguientes desplazamientos, en los que desaparecen un gran número de piezas sobre el tablero, el número de posibles movimientos descenderá de igual modo, de ahí el descenso de los valores de tiempo y nodos.



Gráfica 6. 12

En esta tabla resumen (tabla 5.3) se puede observar cómo, al compararla con el modo de juego BOU vs Jugador, se obtienen similares resultados en cuestión de tiempo de respuesta y número de nodos expandidos.

Algoritmo	Profundidad	Media tiempo(sg)	Media nº nodos
Minimax	1	0,02811954	1
	2	2,20541509	87,6363636
	3	93,2943996	2043,8
Alfa-beta	1	0,02045992	1
	2	1,10620807	77,5757576
	3	11,5782191	334,6

Tabla 5.3

El tiempo de respuesta aumenta según aumenta la profundidad, así como el número de nodos expandidos, y estos valores siempre son inferiores en el caso del algoritmo Alfa-beta, gracias a las ventajas que presenta frente al algoritmo Minimax.

5.5. Partidas ganadas

En este apartado se hace un pequeño análisis de las partidas ganadas y perdidas por BOU, en diferentes situaciones de juego.

5.5.1. Fase 1. Movimiento al azar.

Esta tabla recoge los resultados obtenidos en la fase 1 de BOU, explicada en el apartado 4.5.1 de este documento. De 10 partidas jugadas, se obtuvieron 0 victorias

	RESULTADOS
Partidas jugadas	10
Partidas ganadas	0
Partidas perdidas	10

Tabla 5.4

5.5.2. Fase 2. Desplazamiento hacia el castillo

Esta tabla recoge los datos obtenidos de jugar 10 partidas contra BOU utilizando el sistema de movimientos explicado en el apartado 4.5.2 de este documento. Se puede observar que los resultados no fueron satisfactorios.

	RESULTADOS
Partidas jugadas	10
Partidas ganadas	0
Partidas perdidas	10

Tabla 5.5

5.5.3. Fase 3. Búsqueda del movimiento óptimo.

Esta tabla recoge los resultados del juego de 10 partidas contra BOU con el sistema de juego explicado en el apartado 4.5.3.

	RESULTADOS
Partidas jugadas	10
Partidas ganadas	0
Partidas perdidas	10

Tabla 5.6

5.5.4. Fase 4. Búsqueda teniendo en cuenta al oponente.

Con el sistema de juego mediante el cual, se tiene en cuenta un movimiento cualquiera del oponente, y maximizando la jugada de BOU (apartado 4.5.4.), los resultados obtenidos tampoco fueron positivos. En la tabla se puede observar que se ganaron 0 partidas.

	RESULTADOS
Partidas jugadas	10
Partidas ganadas	0
Partidas perdidas	10

Tabla 5.7

5.5.5. Alfa-beta a profundidad 1

Se ha escogido el algoritmo Alfa-beta por sus mejores resultados. En este caso, para obtener resultados más fiables y significativos, se aumenta el número de partidas jugadas a 100. En la tabla a continuación se muestran los datos obtenidos.

	RESULTADOS
Partidas jugadas	100
Partidas ganadas	23
Partidas perdidas	77

Tabla 5.8

5.5.6. Alfa-beta a profundidad 2

Aumentado la profundidad del árbol debería aumentar su destreza y mejorar a su vez su juego. Tras jugar 100 partidas, se muestran ciertos cambios en los resultados:

	RESULTADOS
Partidas jugadas	100
Partidas ganadas	37
Partidas perdidas	63

Tabla 5.9

Han aumentado el número de partidas ganadas lo que confirma que un aumento en la profundidad del árbol de búsqueda, mejora el juego de BOU.

5.5.7. Alfa-beta a profundidad 3

Se procede en esta ocasión a representar los resultados obtenidos de jugar 100 partidas contra BOU haciendo uso del algoritmo Alfa-beta a profundidad 3, se muestran en la tabla a continuación.

	RESULTADOS
Partidas jugadas	100
Partidas ganadas	67
Partidas perdidas	33

Tabla 5.10

Los resultados obtenidos de estas pruebas son significativamente mejores que el resto, ya que las partidas ganadas por BOU suponen más de la mitad de las jugadas, llegando casi a alcanzar el 70 % de victorias.

5.5.8. Alfa-beta a profundidad 4

Por último se ha procedido a probar el juego con el algoritmo Alfa-beta a profundidad 4. Como hemos podido observar en el apartado anterior 5.2.1.4 el tiempo de partida para esta profundidad es notablemente alto, por lo que se ha reducido el número de partidas jugadas a 10. Y los resultados obtenidos se recogen en la siguiente tabla.

	RESULTADOS
Partidas jugadas	10
Partidas ganadas	8
Partidas perdidas	2

Tabla 5.11

Esto supone un 80 % de victorias y por tanto, queda demostrado que el algoritmo a profundidad 4 mejora el juego de BOU respecto al jugado a profundidad 3

5.6. Conclusiones

A modo de resumen, este capítulo recoge las diferentes pruebas realizadas sobre el programa de BOU.

Se ha partido de situaciones menos complejas en las que participan un menor número de piezas en una posición cercana al castillo a conquistar, ejecutando la misma disposición para diferentes algoritmos, e incrementando la profundidad del árbol de búsqueda. No se muestran diferencias significativas hasta la profundidad de 4.

Posteriormente se han realizado pruebas con todas las piezas en juego, para los dos algoritmos y enfrentando a BOU contra el usuario, y se ha demostrado a través de gráficas el comportamiento del tiempo de respuesta, el número de nodos expandidos y la evolución de la función de evaluación, que Alfa-beta es más eficiente que Minimax, aunque en este caso las diferencias ya se muestran a profundidad 3, con tantas piezas en juego, los algoritmos a profundidad 4, pueden prolongar el tiempo de una partida a casi

una semana. Del mismo modo se realizaron pruebas enfrentando a un algoritmo contra otro, con el objetivo de encontrar un comportamiento diferente de BOU; las graficas son más claras, pero presentan comportamientos similares a las partidas del usuario contra BOU

Para terminar, los últimos experimentos analizan el número de partidas ganadas por BOU, partiendo de un mismo número de partidas jugadas, para las diferentes situaciones de las primeras fases programadas (capítulo 4.6), y para las diferentes profundidades del algoritmo Alfa-beta, escogido como el mejor de los dos comparados. Al comienzo no se obtuvo ninguna victoria por parte de BOU, pero a medida que aumentamos la profundidad, mejoramos su número de partidas ganadas.

Capítulo 6. Conclusiones y líneas futuras

En este capítulo se incluye una breve descripción de las conclusiones obtenidas de los resultados del capítulo 5, y los posibles futuros adelantos para BOU

6.1. Conclusiones

Durante la creación y el desarrollo del programa de BOU, partiendo del estado inicial y pasando por cada una de sus fases, se han mostrado diferentes comportamientos y una evolución significativa en su respuesta positiva.

Se han realizado una serie de pruebas de diferente ámbito por si existían factores no contemplados que pudieran influir en el comportamiento de BOU, como fueron los experimentos en los que se comprobaba su respuesta en función del color de piezas escogido y se llegó a la conclusión de que los resultados, independientemente del color que escoja el usuario para jugar contra Bou (blancas o negras), no se ve influido por el numero de nodos expandidos ni el tiempo de decisión; la medias apenas se modifican.

La comparativa entre los dos algoritmos principales que se han escogido y utilizado en las pruebas de BOU, demuestra como ya es sabido, la mayor eficacia del algoritmo Alfa-beta y mayor rapidez gracias a la poda que no incluye el algoritmo Minimax. Diferentes pruebas dieron como resultado un mejor comportamiento por parte de Minimax, frente al usuario, pero no respecto a tiempo de respuesta, sino más bien a la hora de escoger el movimiento más adecuado, pero, dado que ambos algoritmos, utilizan el mismo sistema de evaluación se llegó a la conclusión que fue fruto de la casualidad.

A medida que avanza el juego, y disminuyen el número de piezas (por las capturas que se produzcan en la partida), disminuyen también el número de nodos expandidos y proporcionalmente el tiempo de búsqueda. Esto es debido a que, a menos piezas, menor movilidad y por tanto, la lista de descendientes será mucho menor.

Para el mismo algoritmo y presentando diferentes profundidades, partiendo de un mismo movimiento inicial, BOU presenta diferentes comportamientos en cuanto a la elección del movimiento; así como también, por supuesto, variaciones notables en cuanto al número de nodos expandidos y al tiempo de espera transcurrido, ya que, a medida que disminuyamos la profundidad del árbol de búsqueda, también decrementa la eficacia de BOU, ya que eliminamos la evaluación de los “siguientes” movimientos en el árbol de búsqueda. Cuanto menos profundicemos, más rápida será la respuesta, pero menos eficaz.

El tiempo y el número de nodos en ocasiones sufren bruscos cambios debido a la influencia de métodos como “hay_capturas”, o “es_batalla_ganada” cuyos efectos ya se han explicado en el apartado 5.1, o de factores como el de la movilidad de las piezas (capitulo 4.5.1.9.). Este tipo de agentes hace que caiga en picado el valor del número de nodos expandidos y del tiempo, dando lugar a datos aislados en los estudios de comportamiento de un algoritmo. Por lo que no se puede hablar de una respuesta homogénea del programa.

Referente a las pruebas a diferente profundidad, se llega a la conclusión de que, a profundidad 1, BOU, no siempre gana, pero presenta bastante competitividad, avanza hacia el castillo y se mueve hacia los flancos, protegiendo sus piezas, tiende a la concentración y a las capturas masivas, lo que nos lleva a pensar que posee una función de evaluación es bastante acertada si a nivel 1 muestra resistencia a ser ganado, pero, obviamente presenta las carencias de la incapacidad de prever el movimiento del contrario; en un movimiento simple de avances, puede llegar a sacrificar piezas, porque no ve la captura de la pieza enemiga frente a la suya

A mayor profundidad, mayor competencia, y cabe destacar, que predominan en sus decisiones, el mismo canon en la mayoría de las partidas: Bou sigue una estrategia ofensiva, no defensiva y su tendencia es la captura de las piezas del enemigo, por encima de la conquista del castillo.

Con factores nuevos que dirijan sus movimientos más directamente al castillo, y técnicas de resolución de situaciones, Bou podría llegar a volverse, invencible.

6.2. Líneas futuras

Quedan muchos flancos abiertos en cuanto al desarrollo del este juego, por tanto, caben diferentes posibilidades, para futuros trabajos, en los que mejorar su efectividad.

6.2.1. Tablas precomputadas:

Ante la situación que se plantea en el desarrollo del juego, se podrían establecer una serie de pistas a seguir con las cuales ante un mismo patrón de repetición, la velocidad de respuesta será menor al tener ya el resultado del movimiento. Ayudará a conocer la resolución y a escoger la mejor opción a la hora de desplazar una pieza, evitando tener que recorrer el árbol de búsqueda en cada una de ellas.

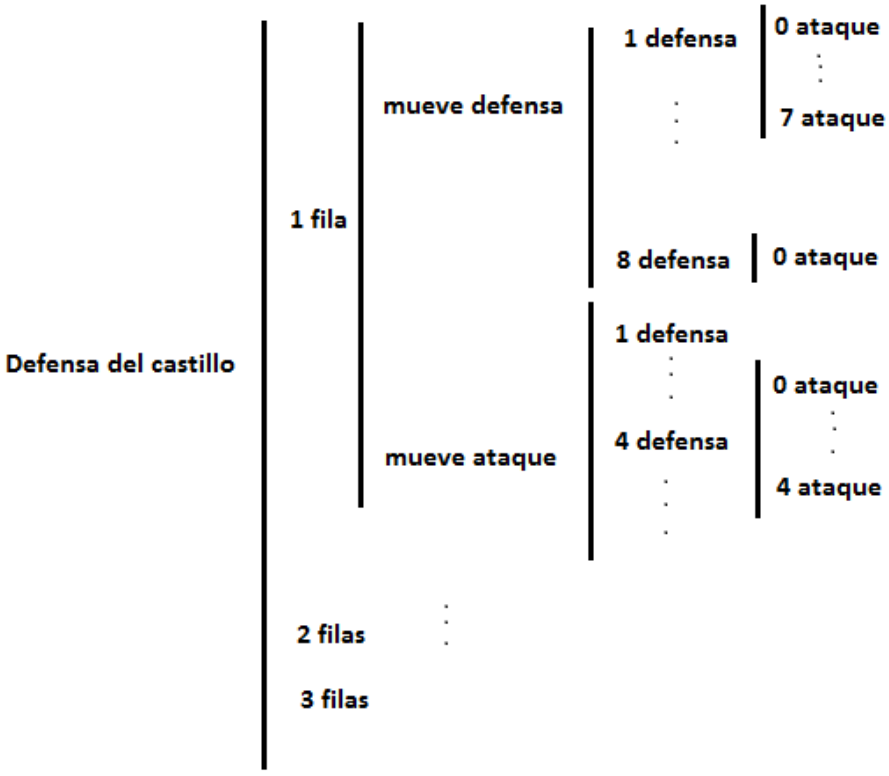
En ocasiones se trataría de una situación ya almacenada, y en otras ocasiones se trataría de una situación en la cual, uno de sus pasos intermedios se ha establecido también como ya almacenado.

Estas tablas podrían guardar todos los posibles movimientos que pueden hacer las piezas teniendo en cuenta las tres filas más próximas a los castillos tanto para el atacante, como para el que defiende.

En este caso se establecen una serie de simetrías que también pueden ahorrar situaciones repetidas en estas tablas precomputadas. Para empezar, se toman solo las filas más cercanas al castillo porque a mayor numero de casillas, mayor número de piezas en juego y finalmente se disparan el número de combinaciones que se podrían producir.

Esquema para las tablas precomputadas

Este sería un borrador del esquema de la estructura que podrían tener las tablas; teniendo en cuenta todas las posibilidades de combinación para la ocupación de las 3 filas más próximas al castillo.



Simetrías e interacciones:

Diferenciar entre atacantes y defensores, en las tablas precomputadas no tiene ningún sentido que se haga distinción de piezas blancas y negras ya que se van a producir una serie de coincidencias en los resultados

Una situación en la que un atacante caballero blanco intenta hacer una intrusión en la que dos hombres negros defienden el castillo, tendrá iguales consecuencias si la situación se ha producido en el otro lado del tablero con un caballero negro como atacante y dos hombres blancos en la defensa.

Con esta medida se consigue reducir en gran medida el tamaño de las tablas precomputadas ya que se pueden obtener simetrías en el tablero tanto en el color de las piezas como en el eje imaginario que separa el tablero de manera vertical.

Aparte de la repetición de las situaciones de la mano de las simetrías producidas en el tablero, también se reduce el tamaño de las tablas precomputadas, teniendo en cuenta que, una situación que se ha originado a una distancia x del castillo en su evolución puede terminar en otra que ya ha sido contemplada en una distancia x-y al castillo; situaciones que se deben tener en cuenta.

6.2.2 Función ranking

Se debería establecer un número para representar diferentes situaciones que se puedan producir durante el ataque o la defensa de cualquiera de los dos jugadores en la partida.

Este número obtenido, dependiendo de la situación y el número de participantes en el análisis, de diferentes fórmulas, recoge:

Tipo de tabla: ataque o defensa
Número de filas
Quién mueve
Número de defensas
Número de atacantes
Posición de las fichas

El resultado

El número obtenido en la función de ranking, aparte de dirigir hacia una posición de las tablas precomputadas, nos proporciona el resultado de esta situación.

Se establecen dos criterios:

1. Signo del resultado:

Se establecería previamente que el resultado será positivo si el atacante ha conseguido alcanzar una de las casillas del castillo y el número resultado será negativo si el atacante ha fallado en su intento y la defensa ha logrado bloquear la intrusión

2. Número de movimientos para el éxito:

El número que se obtiene en la casilla correspondiente de la tabla precomputada nos indica el número de movimientos que se van a llevar a cabo hasta que la situación finalice. Ya sea por un bloqueo o por una entrada del ataque al castillo.

6.2.3. Factores de la función de evaluación

Teniendo en cuenta la situación actual del programa, las modificaciones más sencillas que pueden mejorar su toma de decisiones sin modificar su estructura, son las que tienen que ver con su función de evaluación. Podrían incluirse nuevos factores que influyeran positivamente sobre ella.

-Distancia al castillo (f_{11}): Es el número de casillas entre tus oponentes y tu castillo. Se van tomando las piezas una a una del tablero y se cuentan las casillas de distancia que le lleven a su objetivo por la vía más rápida. Es una acumulación de la distancia al destino de todas las piezas. Aquel jugador que posea un menor número de esta distancia, quiere decir que posee sus piezas en general más cerca del castillo enemigo se encuentra en una situación de ventaja frente al contrario, que sus piezas estuvieran a una distancia muy alejada del castillo oponente.

-Contador de capturas: En la lista de posibles movimientos, evaluar aquellas capturas múltiples que capturen un mayor número de piezas enemigas.

-Búsqueda de la carga del caballero

-Ataque preventivo: cuando se localice una posibilidad de una captura a una distancia de uno, localizarla.

-Protección de piezas: en cuanto una pieza es vulnerable a ser comida, toma una de las dos posibilidades; o bien retroceder para evitar ser comida, o bien, posicionar una pieza amiga en la casilla destino para evitar la captura.

6.2.4. Mejora en la interfaz

Se pueden incluir nuevas opciones en la interfaz que mejoren el juego del usuario. Un ejemplo de ello, es un botón que, al pulsarlo, le proporcione al jugador en turno una sugerencia de movimiento en caso de no saber cuál es el movimiento más adecuado para realizar.

6.2.5. Adaptación a la tecnología móvil

Modificando el código para poder crear una aplicación móvil con el programa, se garantiza su distribución y su uso a nivel masivo, dado que la tecnología móvil supone hoy en día la herramienta personal de bolsillo de casi cualquier persona.

Capítulo 7. Planificación y presupuesto

A continuación se plantea un desglose de la planificación del desarrollo del proyecto de creación del juego de Camelot. Se ha divido este desarrollo en tres partes para poder hacer distinción entre tareas y tiempo, recursos, humanos y material, y finalmente, costes derivados de los dos anteriores.

7.1. Planificación de tareas

Se plantea en primer lugar un esquema y planificación de las tareas inicialmente tomadas en cuenta, al comienzo del proyecto, así como una estimación inicial realizada, de los tiempos que tomaría llevarlos a cabo. Posteriormente se plantea un esquema de las tareas reales realizadas así como el tiempo real que ha tomado cada una de ellas. Se pueden observar ciertas diferencias, sobretodo en la tarea de desarrollo de la aplicación, que los tiempos han sido distintos a como se planteaba inicialmente.

Se muestra el desarrollo de las tareas mediante un diagrama de Gantt.

7.1.1. Estimación inicial

En primer lugar se muestra un listado de las tareas así como una estimación temporal de cada una de ellas:

id	Nombre Tarea	Tiempo (sem)	Fecha inicio	Fecha fin
1	Planificación tareas	2 semanas	6/12/2010	20/12/2010
2	Especificación de requisitos	1 semana	21/12/2010	28/12/2010
3	Análisis y diseño del producto	2 semanas	29/12/2010	14/01/2011
4	Desarrollo del código	12 semanas	17/01/2011	11/04/2011
5	Investigación de IA del juego	2 semanas	12/04/2011	25/04/2011
6	Desarrollo de IA en el código	5 semanas	26/04/2011	31/05/2011
7	Pruebas	3 semanas	1/06/2011	22/06/2011
8	Creación de la interfaz	4 semanas	23/06/2011	22/07/2011

id	Nombre Tarea	Tiempo (sem)	Fecha inicio	Fecha fin
9	Experimentos y resultados	2 semanas	25/07/2011	05/08/2011
10	Documentación	35 sem (8 sem)	6/12/2011	30/09/2011
11	Seguimiento	35 sem(70 h)	6/12/2010	30/09/2011

Tabla 7.1

En la figura 8.1 se puede observar la disposición de las tareas, identificadas por su id, en un diagrama Gantt, para conocer el inicio y fin de cada una de ellas y el orden de realización.

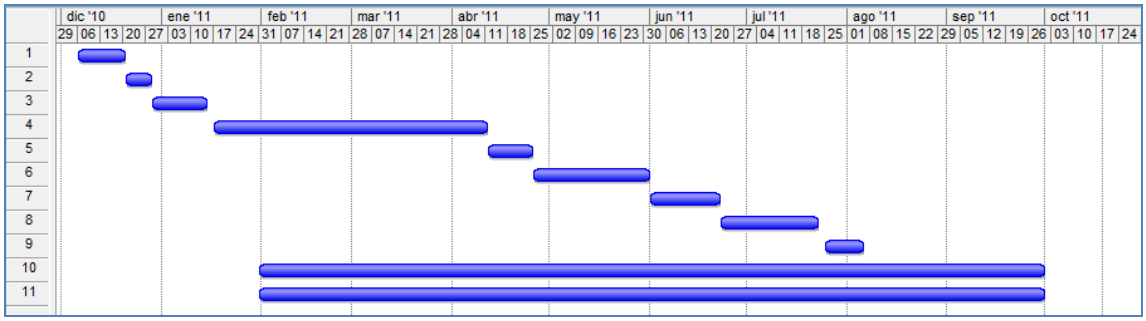


Figura 7. 1

7.1.2. Planificación real

A continuación se detalla la mis tabla de tareas, pero con la duración y fechas reales de cada una de ellas

Nombre Tarea	Tiempo (sem)	Fecha inicio	Fecha fin
Planificación tareas	2 semanas	6/12/2010	20/12/2010
Especificación de requisitos	1 semana	21/12/2010	28/12/2010
Análisis y diseño del producto	2 semanas	29/12/2010	14/01/2011
Desarrollo del código	14 semanas	17/01/2011	25/04/2011
Investigación de IA del juego	1 semanas	26/04/2011	04/05/2011
Desarrollo de IA en el código	5 semanas	05/05/2011	09/06/2011

Nombre Tarea	Tiempo (sem)	Fecha inicio	Fecha fin
Pruebas	6 semanas	10/06/2011	22/07/2011
Creación de la interfaz	3 semanas	22/07/2011	12/08/2011
Experimentos y resultados	2 semanas	15/08/2011	29/08/2011
Documentación	35 semanas (7 semanas)	6/12/2011	28/10/2011
Seguimiento	35 semanas(70 horas)	6/12/2010	28/10/2011

Tabla 7.2

Como se puede observar en el diagrama de Gantt de la figura 8.2, se ha producido un retraso en determinadas tareas, por lo que se ha recurrido a la disminución de tiempo de otras de menos importancia, aún así se ha prolongado la fecha de cierre del proyecto.

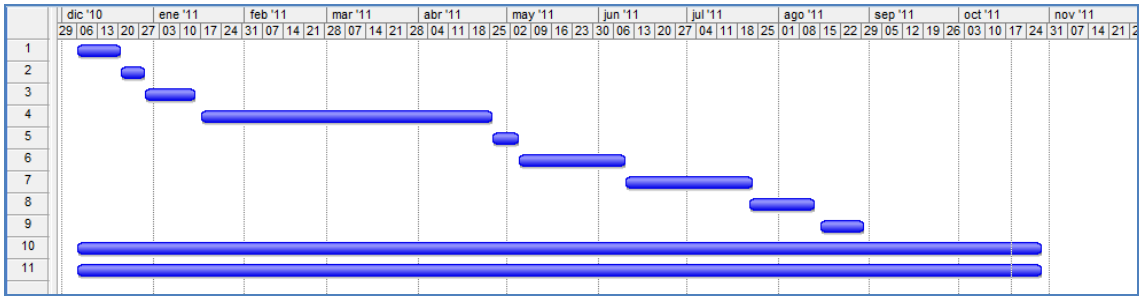


Figura 7. 2

7.2. Panificación de recursos

A continuación se muestra un listado de los recursos utilizados en el desarrollo del proyecto, separados en software y hardware.

7.2.1. Hardware

- PC Asus Intel(R) Atom (TM), 1.76GHz, 2Gb RAM, Windows 32 bits
- PC Intel Core 2 Duo, 2.67 GHz, 6 Gb RAM, Windows 64 bits

7.2.2. Software

-Sistemas operativos:

-Microsoft Windows XP

-Microsoft Windows 7

-Apple R Mac TM OS X Leopard

-Programas desarrollo código java:

- Eclipse helios

-Eclipse Galileo

-Netbeans IDE 7.0

-Lenguajes de programacion:

- Java TM (Sun Microsystems) 1.7

-Procesador de Textos:

-Microsoft Word 2007

-Dziagramas:

-Visual Paradigm

-Herramientas de dibujo y graficas:

-Microsoft Excel 2007

-GIMP 1.6

-GIMP 1.7

-Paint

-Presentation:

-Microsoft PowerPoint 2007

7.3. Análisis económico

En este apartado se presente mostrar un análisis económico del proyecto llevado a cabo.

7.3.1. Recursos

Para una mayor claridad en el desglose de los costes económicos, se hacen tres pequeños grupos con los recursos para mostrar el análisis económico de cada uno de ellos. Los grupos son los siguientes:

- Recursos humanos: en este grupo, formado por dos individuos, uno con el rol de programador y el segundo con el rol de supervisor y director de proyecto, se estiman los costes en función de los salarios aproximados de ambos roles

- Recursos hardware: Los recursos que componen este grupo son los equipos especificados en el apartado 7.2.1 de hardware usado en el proyecto, se tiene en cuenta el coste de adquisición del equipo, y su devaluación a lo largo del tiempo, por tanto el coste corresponde con su precio mensual obtenido de dividir la inversión inicial por el número de mese estimados de duración, multiplicado por el número de meses de uso en el proyecto.
- Recursos Software: forman parte de este grupo, todos los recursos incluidos en el apartado 7.2.2 del software utilizado en este proyecto y se tienen en cuenta la adquisición de las licencias de los mismos en el periodo de desarrollo del producto.

7.3.2. Presupuesto

En cuanto a los recursos personales, hay que aclarar, que del tiempo total de proyecto, que han sido 45 semanas, hay que descartar fines de semana, festivos y dos semanas de vacaciones. Para el programador, en las fases de análisis, desarrollo y especificación de requisitos, las horas semanales invertidas han sido de 15, mientras que en el resto del proyecto, subieron a 40 horas semanales. Las primeras semanas suman 60 horas y el resto del proyecto toman llevan 1520 horas, lo que hace un total de 1580 horas

Las tareas de supervisión por parte del director de proyecto se reducen a 2horas semanales, lo que hacen un total de 90 horas

Recursos personales	Horas invertidas	Precio/hora	Coste total
Programador	1580 horas	20€ /hora	31600 €
Director de proyecto	90 horas	27€/hora	2430 €
			34030 €

Tabla 7.3

Recursos hardware	Inversión	Tiempo de vida	Tiempo uso	Coste total
PC Asus	399€	36 meses	6 meses	66,49 €
PC Intel Core 2Duo	1200€	60 meses	3 meses	60 €
				126,49 €

Tabla 7.4

Recursos Software	Coste(€)	Tiempo de vida	Tiempo de uso	Coste total
Microsoft Windows 7	300	36 meses	9 meses	75 €
Eclipse	10	12 meses	5 meses	4,17 €
Netbeans IDE 7.0	10	12 meses	2 meses	1,67 €
Visual Paradigm	150	12 meses	1 meses	12,5 €
GIMP 1.7	10	12 meses	3 meses	2,5 €
				95,84 €

Tabla 7.5

En definitiva, si juntamos los costes personales, los costes de hardware y los costes de software, obtenemos el total del coste del proyecto, que se muestra en la tabla a continuación.

Concepto	Coste (€)
Recursos personales	34030 €
Recursos Hardware	126,49 €
Recursos software	95,84 €
Coste total del proyecto	34252,33 €

Tabla 7.6

Capítulo 8. Apéndices

Este capítulo incluye documentación que puede ser de interés para la comprensión de ciertos conceptos incluidos en el resto de capítulos de esta memoria.

8.1. Algoritmos de búsqueda:

En este apartado se presentan los códigos de los algoritmos explicados en el capítulo 2 de esta memoria y utilizados en el desarrollo de la programación de Camelot.

8.1.1. MiniMax:

A continuación se muestra el código del algoritmo Minimax, aplicado al desarrollo del juego de Camelot. Para más información consultar el apartado 2.

```
VecTemp Minimax(int ply, Tablero t, VecTemp movaux, boolean jug,boolean col,
int prof, Nodo n){

	VecTemp scor= new VecTemp();
	VecTemp fin= new VecTemp();
	VecTemp[] L= new VecTemp[maxmax];

	if (fin_partida(t,false)||(prof==0)){
		double z=Math.ceil((ply*1.0)/2);
		fin=movaux;
		fin.score=t.funcion_evalua((int)z,!col, false);
		return fin;
	}
	L=t.hallar_descendientes(col);
	n.incr_nodo();
	if (t.hay_capturas(L)){
		L=t.escoger_capturas(L);
	}

	if (jug==true){
		fin.score=-10000;
	}else{
		fin.score=10000;
		}
	int j=0;
	while (L[j]!=null){

		Tablero tabaux=t.movim_tablero(L[j],col);
		scor=Minimax(ply+1,tabaux,L[j],!jug,!col,prof-1,n);

		if ((jug==true)&&(scor.score>fin.score)){
			fin=L[j];
			fin.score=scor.score;
		}
		if ((jug==false)&&(scor.score<fin.score)){
			fin=L[j];
			fin.score=scor.score;
		}
		j++;
	}
	return fin;
}
```

8.1.2. Negamax:

Se muestra el pseudocódigo del algoritmo Negamax, debido a que éste no ha sido aplicado al desarrollo del juego de Camelot, pero, de igual modo, es significativo para la introducción a los algoritmos de búsqueda. Para conocerlo en más profundidad, visitar el apartado 2.3

```
float negamax (VecTemp p) {

      integer i,w;
      float m,t;

      if(es_nodo_terminal(p)){

            m=funcion_evaluacion(p);
            return m;

      }

      W= generar_arbol(p);

      m= -∞;

      for (i=0; i=w; i++){

            t=-negamax(pi);

            if(t>m){
                  m=t;
            }
      }
      return m;
}



}
```

* Aclaración:, Con el algoritmo Negamax ka función de evaluación se aplica para el jugador al que le toca mover en el nodo p, y no para aquel para quién se esta resolviendo el árbol de búsqueda cómo sucede con Minimax

8.1.3. Alphabeta

Se muestra a continuación el código del algoritmo Alfa-beta utilizado en el desarrollo del juego de Camelot; para una explicación mas detallada, consultar el apartado 2. de este documento.

```
VecTemp alphabeta (int ply, Tablero t, VecTemp movaux, boolean jug, boolean
col, float alpha, float beta, int prof, Nodo n){

	VecTemp fin= new VecTemp();
	VecTemp scor= new VecTemp();
	VecTemp[] L= new VecTemp[maxmax];

	if (fin_partida(t,false)||(prof==0)){
		double z=Math.ceil((ply*1.0)/2);
		fin=movaux;
		fin.score=t.funcion_evalua((int)z,!col, false);
		return fin;
	}
	L=t.hallar_descendientes(col);
	n.incr_nodo();
	if (t.hay_capturas(L)){
		L=t.escoger_capturas(L);
	}

	if (jug==true){
		fin.score=Math.max(alpha,-1000);
	}else{
		fin.score=Math.min(beta,1000);
		}
	int j=0;
	while (L[j]!=null){

		Tablero tabaux=t.movim_tablero(L[j],col);

		if (jug==true){
			scor= alphabeta(ply+1,tabaux,L[j],!jug,!col,
		Math.max(alpha,fin.score),beta,prof-1,n);
		}else{
			scor=alphabeta(ply+1, tabaux, L[j],!jug,!col,alpha,
		Math.min(beta,fin.score),prof-1,n);
		}

		if ((jug==true)&&(scor.score>fin.score)){
			fin=L[j];
			fin.score=scor.score;
		}
		if ((jug==false)&&(scor.score<fin.score)){
			fin=L[j];
			fin.score=scor.score;
		}
		if ((jug==true && fin.score>=beta)||(jug==false &&
		fin.score<=alpha)){

			return fin;
		}

		j++;
	}
	return fin;
}
```

8.1.4. F Alphabeta

Se muestra a continuación el código correspondiente al algoritmo de Falfa-beta utilizado en el desarrollo del juego de Camelot; para mas información, visitar el apartado 2.

```
VecTemp Falphabeta (int ply, Tablero t, VecTemp movaux, boolean jug,
boolean col, float alpha, float beta, int prof, Nodo n,boolean
cambio){

	VecTemp fin= new VecTemp();
	VecTemp scor= new VecTemp();
	VecTemp[] L= new VecTemp[maxmax];

	if (fin_partida(t,false)||(prof==0)){
		double z=Math.ceil((ply*1.0)/2);
		fin=movaux;
		fin.score=t.funcion_evalua((int)z,!col, false);
		if (cambio){fin.score=-fin.score;}
		return fin;
	}

	L=t.hallar_descendientes(col);
	n.incr_nodo();
	if (t.hay_capturas(L)){
		L=t.escoger_capturas(L);
	}

	fin.score=-10000;

	int j=0;
	while (L[j]!=null){
		double z=Math.ceil((ply*1.0)/2);
		float ev=t.funcion_evalua((int)z,!col, false);
		Tablero tabaux=t.movim_tablero(L[j],col);
		cambio=true;
		alpha=ev-100;
		beta=ev+100;
		scor= Falphabeta(ply+1,tabaux,L[j],jug,!col,-beta,-
		Math.max(alpha,fin.score),prof-1,n,cambio);

		if (scor.score>fin.score){
			fin.score=scor.score;
			fin.movim=L[j].movim;
		}

		if (fin.score>=beta){
			fin.movim=L[j].movim;
			return fin;

		}
		j++;
	}
	if (cambio){fin.score=-fin.score;}
	return fin;
}
```

8.2. Técnicas de aplicación

En este apartado se muestran los códigos de las técnicas de aplicación sobre los algoritmos tratados en el capítulo 2 de esta memoria.

8.2.1. Profundización iterativa

Para una explicación más detallada del algoritmo de esta técnica, visitar el apartado 2. de este documento

```
float profundizacion_iterativa(Vectemp p; integer tiempo ){

      float m;
      integer  d=1; //inicializamos a profundidad inicial

      While (tiempo>0){
            m= …  //algoritmo escogido para aplicar a profundidad d
            d++;
      }

      Return m;

}
```

8.3. Factores de la función de evaluación

A continuación se presenta el código de todos y cada uno de los factores que componen la función de evaluación del programa de Camelot.

8.3.1. Factor 1 - Número de piezas

Este es el código del factor que contabiliza el material sobre el tablero, para más información consultar el apartado 4.5.1.1.

```
float cuenta_piezas(boolean color){

		float num=0;

		for(int i=0;i<16;i++){
			for (int j=0;j<12;j++){
			    if (tablero[i][j]!=null){
			      Pieza p= tablero[i][j].get_pieza();
			     if(p!=null){
			       if (p.get_color()==color){
					num++;
			       }
			     }
			    }
			}
		}

		return num;
	}
```

8.3.2. Factor 2 – Seguridad

Este factor calcula la seguridad de las piezas sobre el tablero para mas información consultar el apartado 4.5.1.2 de esta documento

```
float calcula_seguridad(boolean jug){
	float segf;
	float bb=0;
	float nb=0;
	float bn=0;
	float nn=0;
	if (jug==false){
		for (int i=0;i<7;i++){	//recorre la territorio de las
negras
			for(int j=0;j<12;j++){
				if (tablero[i][j]!=null){ //si la casilla
existe
					Pieza p= tablero[i][j].get_pieza();
					if(p!=null){  //si hay pieza
						if (p.get_color()==true){
							bn++;
						}else if (p.get_color()==false){
								nn++;
							}
					}//fin del si hay pieza
				}//fin del si existe
			}//fin recorre las columnas
		}
		if (nn==0 && bn==0){
			segf=0;
		}else{
			segf=((nn-bn)/nn+bn+1);}
	} else {//fin de que calculemos negras
		for (int i=9;i<16;i++){	//recorre la territorio de las
blancas
			for(int j=0;j<12;j++){
				if (tablero[i][j]!=null){ //si la casilla
existe
					Pieza p= tablero[i][j].get_pieza();
					if(p!=null){  //si hay pieza
						if (p.get_color()==true){
							bb++;
						}else if (p.get_color()==false){
								nb++;
							}
					}//fin del si hay pieza
				}//fin del si existe
			}
		}//fin recorre las filas
		if (bb==0 && nb==0){
			segf=0;
		}else{
			segf=((bb-nb)/bb+nb+1);}
	} //fin de que sea el jugador blanco


	return segf;
}
```

8.3.3. Factor 3 - Número de caballeros

Se muestra a continuación el código del factor encargado de contar los caballeros en juego para más información consultar el apartad o4.5.1.3.

```
public float cuenta_caballeros(boolean jug){
            float cabf;
            float c=0;
            for(int i=0;i<16;i++){
                  for (int j=0;j<12;j++){
                      if (tablero[i][j]!=null){
                        Pieza p= tablero[i][j].get_pieza();
                       if(p!=null){
                         String clase=p.getClass().getName();
                         if(clase=="Caballero"){
                               if (p.get_color()==jug){
                                     c++;
                               }
                         }
                       }
                     }
                  }
            }
            cabf=c;
            return cabf;
      }
```

8.3.4. Factor 4 - Proximidad al castillo

Aquí se presenta el código del factor de la función de evaluación encargado de calcular la proximidad de las piezas al castillo oponente. Para más información sobre este factor, consultar el apartado 4.5.1.4.

```
float prox_al_castillo(boolean jug){
	float numcas=21000;
	float npiez=0;
	for(int i=0;i<16;i++){ //recorre las filas del tablero
		for(int j=0;j<12;j++){  //recorre las columnas del tablero
			if (tablero[i][j]!=null){ //si la casilla existe
				 Pieza p= tablero[i][j].get_pieza();
				if(p!=null){  //si hay pieza
				  if (jug==true){
					if(p.get_color()==jug){// es blanca
						int aux=i;
						while(aux>0){
							numcas=numcas-100;
							aux--;
						}
					}
				  }else{ //bou juega con las piezas negras
					if(p.get_color()==jug){ //es negra
						int aux=i;
						while(aux<15){
							numcas=numcas-100;
							aux++;
						}
					}

	}
				}//fin contiene pieza
			}//fin la casilla existe
		}//fin de recorrer columnas
	}//fin de recorrer filas
	npiez=cuenta_piezas(jug);
	numcas=numcas-(1400*(14-npiez));

	return numcas;


}
```

8.3.5. Factor 5 - Proximidad a los flancos

A continuación se presenta el código del factor encargado de comprobar los flancos del tablero. Para más información sobre este factor consultar el apartado 4.5.1.5.

```
float prox_al_castillo(boolean jug){
      float numcas=21000;
      for(int i=0;i<16;i++){ //recorre las filas del tablero
            for(int j=0;j<12;j++){  //recorre las columnas del tablero
                  if (tablero[i][j]!=null){ //si la casilla existe
                        Pieza p= tablero[i][j].get_pieza();
                       if(p!=null){  //si hay pieza
                         if (jug==true){ //si comprobamos blancas
                               if(p.get_color()==jug){ //si es blanca
                                     int aux=i;
                                     while(aux>0){
                                           numcas=numcas-100;
                                           aux--;
                                     }
                               }
                         }else{ //si se juega con negras
                               if(p.get_color()==jug){ //si es negra
                                     int aux=i;
                                     while(aux<15){
                                           numcas=numcas-100;
                                           aux++;
                                     }
                               }

      }
                        }//fin contiene pieza
                  }//fin la casilla existe
            }//fin de recorrer columnas
      }//fin de recorrer filas
      return numcas;
}
```

8.3.6. Factor 6 - Pelotones de piezas

Es el código que comprueba cuando pelotones forman las piezas sobre el tablero. Para más información, consultar el apartado 4.5.1.6.

```
float pelotones_piezas(boolean jug){
  float pelotonf;
  float pel=0;
  for (int i=0;i<16;i++){    //recorre la territorio de las negras
    for(int j=0;j<12;j++){
       if (tablero[i][j]!=null){ //si la casilla existe
          Pieza p= tablero[i][j].get_pieza();
          if(p!=null){  //si hay pieza
            if (p.get_color()==jug){//si es del color que evaluamos
              if((i+1)<16 && (j+1)<12){ //si no se sale del rango
                  if (tablero[i][j+1]!=null && tablero[i+1][j]!=null &&
                  tablero[i+1][j+1]!=null){
                      Pieza p1= tablero[i][j+1].get_pieza();
                      Pieza p2= tablero[i+1][j].get_pieza();
                      Pieza p3= tablero[i+1][j+1].get_pieza();
                      if (p1!=null && p2!=null && p3!=null){
                        if( p1.get_color()==jug && p2.get_color()==jug
                              && p3.get_color()==jug){
                              pel++;
                         }
                      }//fin del que comprueba las 3 piezas
                   }//fin del si existen las tres piezas
               }//fin de comprobarel rango
            }//fin de evaluar el color

           }//fin de , si tiene pieza
         }//fin del, si la casilla existe
       }//fin del for de columnas
    }//fin for de filas
    pelotonf=pel;
    return pelotonf;
}
```

8.3.7. Factor 7 - Castillo ocupado

Este factor se encarga de comprobar si el castillo una pieza propia ocupa el castillo oponente. Para más información, consultar el apartado 4.5.1.7.

```
public float castillo_ocupado(boolean jug){
	float cf;
	float c=0;
	if (jug=true){
		if (tablero[0][5]!=null && tablero[0][6]!=null){
			Pieza cb1=tablero[0][5].get_pieza();
			Pieza cb2=tablero[0][6].get_pieza();
			if (cb1!=null)
				if( cb1.get_color()==jug){
					c++;
				}
			if (cb2!=null){
				if(cb2.get_color()==jug){
					c++;
				}
			}
		}
	}else{//fin de que evaluemos blancas
		if (tablero[15][5]!=null && tablero[15][6]!=null){
			Pieza cn1=tablero[15][5].get_pieza();
			Pieza cn2=tablero[15][6].get_pieza();
			if (cn1!=null){
				if(cn1.get_color()==jug){
					c++;
				}
			}
			if (cn2!=null){
				if(cn2.get_color()==jug){
					c++;
				}
			}
		}
	}//fin de que evaluemos negras
	cf=c;

	return cf;
}
```

8.3.8. Factor 8 - Concentración de piezas

Controla cuán concentradas están las piezas sobre el tablero, para más información acudir al apartado 4.5.1.8

```
float concentracion(boolean jug){
       float concf;
       float c=0;;
       if (jug=true){ //evaluamos blancas
        for (int i=1;i<7;i++){    //recorre la territorio de las negras
              for(int j=0;j<12;j++){
                  if (tablero[i][j]!=null){ //si la casilla existe
                     int k=0;
                     Pieza p= tablero[i][j].get_pieza();
                     if (p!=null){
                            if (p.get_color()==jug){
                                   k++;
                                   if (tablero[i][j+1]!=null){
                                     Pieza p1= tablero[i][j+1].get_pieza();
                                     if (p1!=null){
                                          if (p1.get_color()==jug){ k++;}}
                                   }
                                   if (tablero[i+1][j]!=null){
                                     Pieza p2= tablero[i+1][j].get_pieza();
                                     if (p2!=null){
                                          if (p2.get_color()==jug){ k++;}}
                                   }
                                   if (tablero[i+1][j+1]!=null){
                                     Pieza p3=tablero[i+1][j+1].get_pieza();
                                     if (p3!=null){
                                          if (p3.get_color()==jug){ k++;}}
                                   }}}//fin de que exista pieza
                     if (k>0){ c=c+((6-i)*k);}
                  }}}//fin recorre filas
       }else{// evaluamos negras
         for (int i=9;i<15;i++){   // territorio de las blancas
              for(int j=0;j<12;j++){
                  if (tablero[i][j]!=null){ //si la casilla existe
                     int k=0;
                     Pieza p= tablero[i][j].get_pieza();
                     if (p!=null){
                            if (p.get_color()==jug){
                                   k++;
                              if (tablero[i][j+1]!=null){
                                   Pieza p1= tablero[i][j+1].get_pieza();
                                   if (p1!=null){
                                          if (p1.get_color()==jug){ k++;}}
                              }//fin evalua pieza contigua 1
                              if (tablero[i+1][j]!=null){
                                   Pieza p2= tablero[i+1][j].get_pieza();
                                   if (p2!=null){
                                          if (p2.get_color()==jug){ k++;}}
                              }//fin evalua pieza contigua 2
                              if (tablero[i+1][j+1]!=null){
                                   Pieza p3= tablero[i+1][j+1].get_pieza();
                                   if (p3!=null){
                                          if (p3.get_color()==jug){ k++;}}
                                   }}}//fin de que la casilla no este vacia
                            if (k>0){ c=c-((8-i)*k); }
                  }}}//fin recorre filas
       }//fin de que evaluemos negras
       concf=c;
       return concf;
}
```

8.3.9. Factor 9 - Movilidad

Aquí se muestran las dos función es encargadas de controlar la movilidad de las piezas. Para más información de ambas, consultar el apartado 4.5.1.9 de este documento

8.3.9.1. movilidad

Esta función se encarga de recorrer el tablero encontrando cada pieza del color que se indica

```
float movilidad(boolean jug){
      float movf;
      float m=0;
      float num;
      for(int i=0;i<16;i++){  //Recorre el tablero en busca de piezas
            for (int j=0;j<12;j++){       //columnas
                if (tablero[i][j]!=null){       //si la casilla existe
                  Pieza p= tablero[i][j].get_pieza();//evalua contenido
                 if(p!=null){      //si contiene pieza
                   if (p.get_color()==jug){
                         num=cuenta_movilidad(i,j);
                         m=m+num;
                   }
                 }
                }
            }
      }
      movf=m;
      return movf;
}
```

8.3.9.2. Cuenta movilidad

Esta función dada una pieza comprueba cuantos movimientos puede realizar.

```
float cuenta_movilidad(int f, int c ){
      float posibles=0;
      Pieza porig=tablero[f][c].get_pieza(); //guardamos origen
      for (int j=0;j=7;j++){//para los 8 posibles movimientos
         if ((casilla-contigua!=null){
            Pieza p= tablero[f][c-1].get_pieza();//evalua que contiene
            if(p==null){                          //si esta vacia
              posibles++; //aumentamos el contador de movilidad
            }else{        //si contiene una pieza
               if ((c-2)>-1 && tablero[f][c-2]!=null){//existe destino
                        Pieza aux= tablero[f][c-2].get_pieza();
                        if(aux==null){           //si esta vacia
                              posibles++;
                        }    //fin del si esta vacia
                  }    //fin de si existe la casilla destino
               }
           }//fin de que haya pieza contigua
      }//fin del bucle de los 8 movimientos
      return posibles;
}
```

8.3.10. Factor 10 - Número de combos caballero-hombre

Este factor es el encargado de comprobar cuantas parejas de caballero y hombre hay formadas sobre el tablero de juego. Para más información consultar el apartado 4.5.1.10 de esta memoria.

```
float busca_combos(boolean jug){
 float combosf;
 float c=0;
 if (jug==true){ //evaluamos blancas ahora, posteriormente negras
  for(int i=0;i<16;i++){
   for (int j=0;j<12;j++){
       if (tablero[i][j]!=null){
        Pieza p= tablero[i][j].get_pieza();
        if(p!=null){
           String clase=p.getClass().getName();
           if(clase=="Caballero"){//si encontramos un caballero
             if (p.get_color()==jug){
                if((i-1)>-1 && (j-1)>-1 && tablero[i-1][j-1]!=null){
                   Pieza cont1=tablero[i-1][j-1].get_pieza();
                   if (cont1!=null){
                      if (cont1.get_color()==jug){//si hay una pieza
                        String clase2=cont1.getClass().getName();
                        if (clase2=="Caballero"){ //si es un caballero
                            c=c+2;
                         }else{c++;}
                      }
                   }
                 }else if((i-1)>-1 && tablero[i-1][j]!=null){
                   Pieza cont2=tablero[i-1][j].get_pieza();
                   if (cont2!=null){
                      if (cont2.get_color()==jug){
                        String clase2=cont2.getClass().getName();
                        if (clase2=="Caballero"){
                            c=c+2;
                         }else{c++;}
                       }
                      }
                  }else if((i-1)>-1 && tablero[i-1][j+1]!=null){
                      Pieza cont3=tablero[i-1][j+1].get_pieza();
                      if (cont3!=null){
                        if (cont3.get_color()==jug){
                          String clase2=cont3.getClass().getName();
                          if (clase2=="Caballero"){
                              c=c+2;
                          }else{c++;}
                        }
                      }
                   }

                  }//fin de que sea del color correspondiente
                 }//fin de que sea un caballero
                }//fin de que haya pieza
               }//fin de que exista la casilla
             } //fin de recorrer columnas
       }//fin de recorrer filas
}
combosf=c;
return combosf;
}
```

8.4. obtener_encadenados

Se presenta a continuación el código de la función con la cual se obtienen los posibles movimientos encadenados de un movimiento dado.

```
public VecTemp[][] obtener_encadenados (Tablero t,VecTemp m,
VecTemp[]pila, int x){
            VecTemp[][] mfaux=new VecTemp[maxmov][movs];
            VecTemp[] lista= new VecTemp[movs];
            VecTemp[][] mf=new VecTemp[maxmov][movs];
            int[] movi=m.movim;
            int f=movi[2];
            int c=movi[3];
            Pieza p=t.tablero[f][c].get_pieza();
            pila[x]=m;
            boolean col=p.get_color();
            lista=busca_movs(t,f,c,p,pila,true);
            if (lista[0]==null){
                  mfaux[0][0]=m;
                  return mfaux;
            }else{
                  int i=0;

                  while(i<movs && lista[i]!=null){
                        Tablero taux=t.copiarTablero();
                        taux.mover_pieza(lista[i].movim,true,col);

      mfaux=obtener_encadenados(taux,lista[i],pila,x+1);
                        mf=unir_listas(mf, mfaux,m);
                        x++;
                        i++;
                  }
            }
            return mf;
      }
```

8.5. Tablas de resultados

Este apartado contiene las tablas con los datos utilizados para obtener las gráficas del capítulo 6 de resultados.

8.5.1. Tabla Minimax

Esta tabla contiene los resultados obtenidos por el algoritmo Minimax en juego e incluye profundidad de árbol de búsqueda (de 1 a 3) y empiece con diferente color de piezas. Para observar las graficas obtenidas de esta tabla, consultar el apartado 6.2 de este documento

Color (BOU)	**Profundidad**	**Tiempo(sg)**	**Nº nodos**	**movimiento**	**Función evalua**
Negras	1	0,014	1	(9 7 , 8 7)	
		0,019	1	(10 5 , 12 5)	
		0,034	10	(2 6, 1 6)	
	1	0.0429	1	5,9 --> 7,7 --> 9,7 --> 9,9 --> 11,9 --> 9,7 --> 11,5	
		0.03699	1	6,3 --> 4,3 --> 6,1	
		0.0296	1	5,3 --> 5,1 --> 7,1	
		0.01297	1	6,1 --> 8,1 --> 10,1 --> 10,3 --> 8,3 --> 10,5 --> 8,5	
		0.008381671	1	6,4 --> 8,4	
		0.017747637	1	5,2 --> 6,2	
		0.018192739	1	5,4 --> 6,3	
		0.01716236	1	6,3 --> 6,1 --> 8,1	

		0.012475537	1	6,2 --> 7,2	
		0.012108011	1	7,1 --> 8,2	
		0.014161856	1	5,5 --> 7,5 --> 9,3 --> 7,1	
		0.019663612	1	7,2 --> 7,0 --> 9,2 --> 7,2 --> 9,0	
		0.01568611	1	8,2 --> 8,0 --> 10,0	
		0.017346	1	5,8 → 7,8 → 9,8	
		0.01474912	1	9,8 → 7,10	
Blancas	1	0.067258831	1	10,2 --> 9,1	
		0.026224934000000002	1	9,1 --> 8,2	
		0.009180858	1	9,3 --> 11,5 --> 11,3	
		0.021165592	1	11,3 --> 10,2	
		0.014643058	1	. 10,2 --> 9,1	
		0.012813492000000001	1	10,9 --> 8,7	
		0.012100713	1	10,8 --> 8,6 --> 8,8	
		0.006755262000000001	1	9,8 --> 7,8 --> 5,10	
		0.012350340000000001	1	8,8 --> 8,6	
		0.006110843	1	8,7 --> 8,5 --> 10,3 --> 12,1	
		0.015402305000000002	1	8,6 --> 7,5	
		0.006844743	1	10,7 --> 10,9	

		0.020367941	1	12,1 --> 11,1	
		0.01717004	1	10,9 --> 9,8	
		0.013888419	1	11,1 --> 10,1	
		0.013000904	1	9,5 --> 9,3 --> 11,5 --> 9,5	
		0.010112924	1	,5 --> 9,3 --> 11,5 --> 9,5	
		0.01421639	1	9,5 --> 9,3 --> 11,5 --> 9,5	
		0.012679846000000002	1	9,5 --> 9,3 --> 11,5 --> 9,5	
		0.009188923	1	9,5 --> 9,3 --> 11,5 --> 9,5	
		0.00879797	1	9,5 --> 9,3 --> 11,5 --> 9,5	
		0.012959043	1	9,5 --> 9,3 --> 11,5 --> 9,5	
		0.011753158000000001	1	9,5 --> 9,3 --> 11,5 --> 9,5	
		0.011967452	1	9,5 --> 9,3 --> 11,5 --> 9,5	
	1	0.06755915	1	10,2 --> 9,1	
		0.026506819	1	9,1 --> 8,2	
		0.025716848	1	10,9 --> 8,7	
		0.010130973000000001	1	9,3 --> 11,5 --> 11,3	
		0.014503651000000001	1	11,3 --> 10,2	
		0.006121596	1	10,4 --> 12,2	

		0.015045915000000002	1	9,4 --> 8,3	
		0.005811676	1	9,5 --> 7,3	
		0.026519492000000002	1	10,5 --> 9,4	
		0.008428908	1	9,6 --> 7,6	
		0.011493163	1	10,6 --> 9,5	
		0.010817252000000001	1	12,2 --> 11,1	
		0.011307288	1	11,1 --> 10,1	
		0.004397641	1	9,8 --> 9,6 --> 11,8 --> 9,8 --> 11,6 --> 13,4	
		0.005356588000000001	1	10,7 --> 8,7	
		0.005352748	1	10,8 --> 8,6	
		0.005447222000000001	1	9,5 --> 9,6	
		0.008000703	1	10,1 --> 9,1	
		0.007850544000000001	1	13,4 --> 12,3	
		0.011820749	1	12,3 --> 11,2	
		0.012775472000000001	1	11,2 --> 10,2	
		0.011621432000000001	1	7,3 --> 7,2	
		0.008015681	1	7,2 --> 7,1	
		0.003255122	1	10,2 --> 8,0 --> 6,2 --> 4,4 --> 6,4 --> 4,6 --> 6,8	
		0.008666245000000001	1	6,8 --> 7,9	

Negras	2	0.157911837	15	5,9 --> 7,7 --> 9,7	
		0.815201912	129	6,6 --> 7,7	
		0.060493583000000004	10	6,8 --> 4,6 --> 6,6 --> 4,8 --> 6,8 --> 8,8	
		1.466085051	180	6,4 --> 7,4	
	2	1.782096136	141	6,8 --> 7,9	
		0.026042899	2	5,8 --> 3,10	
		1.449632397	131	6,4 --> 7,3	
		0.012695592	2	6,7 --> 6,9	
		0.376189756000000004	51	6,5 --> 7,6	
		0.129517846000000002	16	6,9 --> 7,8	
		0.102951117000000001	15	3,10 --> 2,10	
Blancas	2	1.273678749	141	9,4 --> 8,4	
		0.015085856000000002	2	10,5 --> 12,3	
		0.054911563000000004	11	10,9 --> 8,7 --> 6,7	
	2	1.241087201	141	9,4 --> 8,4	
		0.012391432	2	10,5 --> 12,3	
		0.052074661	11	10,9 --> 8,7 --> 6,7	
		0.717403844	111	10,7 --> 8,5	
		0.014353107	2	9,5 --> 9,3	
		0.277571761000000003	65	9,3 --> 8,4	

		0.023026649	3	8,4 --> 8,2	
		0.41059243500000003	57	9,7 --> 8,6	
Negras	3	1.4164624860000001	172	5,9 --> 7,7 --> 9,7 --> 9,9 --> 11,9 --> 9,7 --> 11,7	
		112.204534666	13758	5,2 --> 6,1	
		153.747222623	18449	6,4 --> 7,4	
		1.089064617	88	6,8 --> 4,6 --> 6,4 --> 4,4 --> 6,2 --> 6,4 --> 8,4 --> 10,2 --> 8,2	
		0.7619904940000001	81	,4 --> 9,6 --> 11,4 --> 9,2	
		45.351962214000004	5662	5,4 --> 5,2 --> 7,4	
		0.2352713010000002	26	6,3 --> 8,5 --> 8,7	
		40.070462875000004	4796	6,1 --> 7,2	
		39.230042214	4607	7,2 --> 7,3	
		44.787188121	4907	7,4 --> 7,2	
		38.548847189	4396	5,8 --> 6,9	
		39.729658174	4750	8,7 --> 8,8	
		0.408111537	52	6,7 --> 8,7	
		16.01999427	2598	5,7 --> 7,5	
		0.005592006	1	8,7 --> 10,9	
	3	1.371362978	172	5,9 --> 7,7 --> 9,7 --> 9,9 -->	

				11,9 --> 9,7 --> 11,7	
		104.412488345	13758	5,2 --> 6,1	
		142.571597163	18449	6,4 --> 7,4	
		0.087008951	13	6,8 --> 4,6 --> 6,4 --> 4,4 --> 6,2 --> 6,4 --> 8,4 --> 10,6 --> 8,6	
		3.885627647	864	6,1 --> 7,0	
		12.273796082	1597	7,4 --> 6,4	
		6.065081209000001	1040	6,3 --> 4,5	
		0.9348427520000001	147	7,0 --> 9,2 --> 11,4 --> 9,4 --> 11,6	
		28.029580063	3853	11,6 --> 12,7	
		27.163592835000003	3699	5,4 --> 6,3	
		46.715985078	4921	5,8 --> 7,6 --> 5,4 --> 5,2 --> 7,4 --> 5,4 --> 7,2	
		0.009733801	1	6,4 --> 8,2 --> 8,0	
Blancas	3	19.773917822	2196	10,3 --> 11,2	
		1.253314623	160	10,2 --> 8,2	
		8.357458920000001	1193	10,9 --> 8,7	
		84.941271531000001	13007	10,8 --> 8,6 --> 8,8	
		1.851282837	285	8,7 --> 10,9 -->	

				8,9	
		130.040708686 00002	17766	11,2 --> 10,1	
		108.719866155	16058	8,9 --> 9,9	
		146.336308084	19460	10,4 --> 8,6 --> 10,8	
		111.842534475 00001	15739	9,8 --> 10,9	
		146.817565927	19738	9,4 --> 11,6	
		104.931049235	15610	9,5 --> 11,5 --> 11,7 --> 9,5	
		105.262161299 00001	15610	9,5 --> 11,5 --> 11,7 --> 9,5	
		105.245810157 00001v	15160	9,5 --> 11,5 --> 11,7 --> 9,5	
		105.416145529	15160	11,6 --> 9,4	
		145.622667552	19432	9,4 --> 11,6 --> 9,8 --> 9,10	
		151.62902874	19887	9,6 --> 9,4 --> 11,6 --> 9,6	
		0.496497583	164	9,6 --> 8,5	
	3	160.502721286	18139	9,3 --> 8,2	
		13.3138465290 00001	2648	. 8,2 --> 9,1	
		326.847494696	31143	10,7 --> 8,5	
		4.40279870800 0001	495	9,7 --> 7,5 --> 5,3 --> 5,1	
		24.0575014710 00002	2683	10,3 --> 9,2	
		0.16867172400 000002	18	10,2 --> 8,2 --> 6,4 --> 4,6	

		196.40582223	22265	10,8 --> 8,8	
		2.007174579	383	9,1 --> 9,3 --> 11,5 --> 9,7 --> 9,9 --> 7,7 --> 5,5 --> 7,5	
		0.6784374000000001	100	,8 --> 7,8 --> 5,8 --> 5,10 --> 7,10 --> 5,8	
		21.367785674	3720	5,1 --> 6,0	
		20.571494626	3645	6,0 --> 7,1	
		22.159053134	3795	9,2 --> 8,1	
		18.802106520000002	3492	8,1 --> 7,2	
		16.586861097	3083	8,8 --> 8,9	
		12.845534571	2591	10,9 --> 9,9	
		11.215694746	2542	7,5 --> 6,4	
		7.312990512000001	1889	9,4 --> 8,4	

8.5.2. Tabla Alfa-beta

Esta tabla muestras los datos por los cuales se han obtenido las graficas del algoritmo Alfa-beta del capítulo 6.2 de este documentos (consultar para más información). Contiene diferentes profundidades y diferente color de piezas para BOU.

Color (BOU)	**Profundidad**	**Tiempo(sg)**	**Nº nodos**	**movimiento**	**Función evalua**
Negras	1	0.040323113	1	5,9 --> 7,7 --> 9,7 --> 9,9 --> 11,9 --> 9,7 --> 11,5	118.607574
		0.034652423	1	6,3 --> 4,3 -->	175.37004

				6,1	
		0.024438193	1	5,3 --> 5,1 --> 7,1	212.27965
		0.022617857	1	7,1 --> 8,1	278.0872
		0.00704439	1	5,2 --> 7,0 --> 9,0	275.9019
		0.012591036000000002	1	9,0 --> 8,1	362.71066
		0.011573338	1	6,8 --> 4,8	334.01334
		0.01238404	1	5,8 --> 6,9	364.70535
		0.011304512000000001	1	4,8 --> 5,8	396.29843
		0.011538391	1	6,9 --> 6,8	431.62772
		0.009998786	1	6,1 --> 7,2	447.6659
		0.011214647000000001	1	7,2 --> 8,2	646.828
		0.009839796000000001	1	5,5 --> 5,3 --> 7,5 --> 5,5	814.8076
		0.010773773	1	5,5 --> 5,3 --> 7,5 --> 5,5	881.597
		0.014186326	1	5,5 --> 5,3 --> 7,5 --> 5,5	924.7728
		0.004858452	1	5,4 --> 4,3	995.10986
		0.008157713	1	5,8 --> 7,6 --> 5,4 --> 3,2 --> 1,2	1100.7671
		0.006537077000000005	1	1,2 --> 3,0	1147.7017

		0.006880022	1	3,0 --> 2,1	1199.7886
		0.007420746000000001	1	2,1 --> 1,2	1251.2404
		0.006360036	1	1,2 --> 1,3	1108.8727
		0.007682276000000001	1	1,3 --> 1,2	1155.4047
		0.005759402	1	1,2 --> 1,3	2096.3755
		0.007086251	1	1,2 --> 3,0	2167.639
	1	0.066524803000000001	1	5,2 --> 6,1	26.666668
		0.024492341	1	6,1 --> 7,2	28.412699
		0.023246526	1	5,9 --> 7,7	29.960556
		0.015575387000000001	1	5,8 --> 7,6 --> 7,8	30.723442
		0.013444366000000001	1	5,4 --> 5,2 --> 7,4 --> 5,4	29.947151
		0.013031526000000002	1	5,4 --> 5,2 --> 7,4 --> 5,4	97.30769
		0.014250844	1	5,4 --> 5,2 --> 7,4 --> 5,4	181.8274
		0.018587773000000002	1	5,4 --> 5,2 --> 7,4 --> 5,4	235.25461
		0.011453134	1	7,8 --> 5,8	262.6996
		0.012216216	1	7,7 --> 5,9	287.83145
		0.012139025000000001	1	5,4 --> 5,2 --> 7,4 --> 5,4	313.3825

		0.007174195	1	6,6 --> 8,8	436.25598
		0.014712072000000001	1	5,9 --> 7,7 --> 9,5 --> 11,3 --> 9,1	673.4465
		0.014189014000000002	1	6,8 --> 4,6 --> 6,6 --> 4,4 --> 6,2 --> 8,2 --> 10,4 --> 10,2	861.0875
		0.009297535000000001	1	8,8 --> 8,6	1118.7179
		0.008794831000000001	1	5,3 --> 4,2	1168.0209
		0.008502194000000001	1	4,2 --> 3,1	1248.3448
		0.007263291000000001	1	3,1 --> 2,1	1295.6439
		0.007230648	1	2,1 --> 1,2	1361.9917
		0.007488721	1	1,2 --> 1,3	1444.3259
Blancas	1	0.062889126	1	10,2 --> 9,1	10.0
		0.02445125	1	9,1 --> 8,2	-21.939
		0.008865109000000001	1	9,3 --> 11,5 --> 11,3	-15.394
		0.022427759000000002	1	11,3 --> 10,2	-22.090902
		0.014344165	1	10,2 --> 9,1	-16.358763
		0.012454319	1	9,1 --> 8,1	-23.970297
		0.012395177	1	8,1 --> 7,1	-29.887741
		0.016253212	1	10,9 --> 8,7	58.649582
		0.010657793	1	8,7 --> 10,9	158.15451

		0.010499955	1	7,1 --> 7,2	175.57333
		0.011194678000000001	1	7,2 --> 7,1	193.93105
		0.017064683	1	7,1 --> 7,2	219.59784
		0.006057031	1	9,8 --> 11,10 --> 13,10	55.076088
		0.031675753	1	10,8 --> 9,8	74.55116
		0.016645698	1	10,9 --> 10,8	79.68118
		0.005468686	1	7,2 --> 5,0	20.92207
		0.014078027000000002	1	5,0 --> 4,0	23.754726
		0.019157300000000002	1	4,0 --> 3,0	23.45364
		0.018436463	1	3,0 --> 2,1	246.21335
		0.014187093000000001	1	2,1 --> 1,2	257.59567
		0.013252731	1	9,5 --> 9,3 --> 11,5 --> 9,5	268.62918
		0.008331298000000001	1	9,5 --> 9,3 --> 11,5 --> 9,5	279.67358
		0.008603196	1	9,5 --> 9,3 --> 11,5 --> 9,5	290.72864
		0.010021829000000001	1	9,5 --> 9,3 --> 11,5 --> 9,5	362.58978
	1	0.062842657000000001	1	10,2 --> 9,1	18.857695
		0.01066893100	1	10,9 --> 8,7 -->	94.76567

		0000001		6,7 --> 4,7 --> 6,9 --> 4,9	
		0.020747597	1	9,1 --> 8,2	127.10803
		0.021831349	1	4,9 --> 3,8	144.57005
		0.012504243	1	3,8 --> 2,7	179.35844
		0.023938945000000003	1	2,7 --> 1,6	196.4078
		0.023452753	1	1,6 --> 1,5	196.51535
		0.02295965	1	1,5 --> 0,6	212.52734
		0.017250940000000003	1	8,2 --> 7,1	229.28018
		0.010536054000000001	1	7,1 --> 6,1	374.54996
		0.010412011	1	6,1 --> 5,1	526.9802
		0.011320258000000001	1	5,1 --> 4,1	569.6022
		0.010587899000000001	1	4,1 --> 3,2	613.3519
		0.022822933	1	3,2 --> 2,3	680.8365
		0.022829845	1	2,3 --> 1,4	700.4962
		0.016820819	1	1,4 --> 1,5	996.0816
Negras	2	6.27007649	131	5,2 --> 7,4	23.512726
		5.8918023680000005	126	5,3 --> 7,3	28.277164
		5.49673411600	125	7,3 --> 5,3 -->	27.403492

		0001		7,5	
		0.085111323	2	6,3 --> 8,5 --> 8,3	105.76877
		: 0.130119863	3	7,4 --> 7,2	107.80461
		5.540756577000001	136	5,4 --> 7,4	127.10332
		4.308628402	126	5,8 --> 7,6	155.35997
		3.870292184	116	5,9 --> 7,7	165.57684
		3.116808727	129	9,11 --> 8,10	432.6968
		0.08612327	2	6,8 --> 8,8 --> 10,10	548.91797
		3.4215975210000003	139	10,10 --> 11,11	625.84454
		2.5848159930000003	132	6,7 --> 8,9	784.30054
		0.06862515000000001	2	8,9 --> 10,7	992.2556
		1.518840505	147	5,7 --> 7,5	1053.0203
		0.056643773	2	5,4 --> 4,3	1219.5577
		0.064616166	2	4,3 --> 3,2	1234.1055
		0.049111143	2	3,2 --> 2,1	1266.6902
		0.07654395800000001	2	5,5 --> 4,4	1061.5732
		0.045202553	2	4,4 --> 3,3	1082.0354
		0.063980542	4	3,3 --> 2,4	1097.112

		0.059124060000000006	3	2,4 --> 1,5	1143.1543
		0.043186666000000005	2	1,5 --> 0,5	1149.7341
		0.011472016000000002	2	0,5 --> 2,3	1738.6176
		0.047074315000000005	2	2,3 --> 1,2	1790.5543
		0.036212054	2	1,2 --> 1,4	2471.2402
		0.109652421	13	5,6 --> 7,4	2607.4895
		0.050613974000000006	2	1,4 --> 0,5	2584.757
		0.013256935000000001	2	0,5 --> 1,4	2177.1877
		0.045902231	2	7,8 -- > 8,9	2171.131
		0.014116752000000002	2	8,9 -- > 9,9	2265.7034
		0.050435975	2	9,9-- > 10,9	2253.2231
		0.008007499000000001	2	10,9-- >11,9	2713.5713
		0.05302343	2	11,9-- >12,10	2981.0293
		0.009374829000000001	2	11,11-- >13,9	3005.3323
		0.048237161	2	12,10-- >14,8	2948.469
		0.015453901	2	13,9-- >14,9	2970.6418
		0.04724800300	2	14,8-- >14,7	3009.3198

		0000004			
		0.017631773	2	14,9-- >13,8	3083.123
		0.035641718	2	6,4-- >6,3	3066.5308
		0.011582265000000001	2	13,8-- >15,6	3136.6558
		0.047034281000000004	2	14,7-- >14,6	3115.1086
		0.048599319	2	14,6-- >15,5	3313.7312
	2	2.024981388	131	5,2 --> 7,4	23.512726
		1.7346383410000001	126	5,3 --> 7,3	28.277164
		1.815400002	125	5,9 --> 7,7	34.165966
		1.374250079	116	5,4 --> 7,6	77.3374
		0.042787841	2	6,3 --> 8,3 --> 10,1	107.931114
		1.4221427580000001	117	5,5 --> 7,5	136.5194
		1.4960923810000002	121	5,6 --> 7,8	163.03535
		1.9669117170000001	143	7,3 --> 8,2	185.40277
		1.789351439	133	6,4 --> 8,6	205.26398
		2.375818714	150	6,5 --> 8,5	230.36307
		2.792493738	150	5,7 --> 7,9	244.61835
		2.0259094380000002	152	5,8 --> 6,9	388.51782

		1.5179152660000002	110	7,9 --> 8,10	411.41135
		1.8258996600000001	131	6,9 --> 8,7 --> 6,5	446.38464
		1.4097523570000001	102	7,8 --> 8,9	468.2309
		0.079181572	4	7,7 --> 5,7 --> 7,9 --> 9,9 --> 11,7 --> 11,9	674.7379
Blancas	2	1.32684629	133	10,2 --> 8,4	7.826304
		1.636440386	126	10,3 --> 8,3	6.4273357
		1.517474707	125	10,9 --> 8,7	9.948982
		1.2582694170000002	116	10,4 --> 8,6	21.740004
		1.3345165380000001	117	10,5 --> 8,5	31.905392
		1.2620366550000002	110	10,8 --> 8,8	100.92314
		1.311104623	110	9,3 --> 7,5	179.19131
		1.4076716450000002	115	9,8 --> 7,6	203.44412
		1.208837715	109	9,5 --> 7,3 --> 9,3 --> 9,5 --> 7,7	250.03162
		1.5474296310000002	138	10,6 --> 9,5	299.9555
		1.216256814	95	10,7 --> 9,8	326.1833
		0.98534506500	78	8,3 --> 7,2	346.9172

		00001			
		1.1865337630000001	91	7,2 --> 7,1	349.5543
		0.954539753	85	7,1 --> 7,2	375.63257
		1.038801859	91	8,4 --> 7,3	375.63257
	2	1.26817766	133	10,2 --> 8,4	
		1.517799888	126	10,3 --> 8,3 --> 8,5	16.039371
		0.191227097	8	8,4 --> 8,6 --> 6,6 --> 4,4 --> 6,2 --> 4,2	160.25485
		0.873319095	123	10,9 --> 8,7	188.9832
		0.030667306000000002	2	9,8 --> 7,6 --> 7,8	202.439
		0.735654671	121	10,8 --> 8,6	238.00043
		0.705113638	112	10,7 --> 8,5	271.54242
		0.5694656720000001	95	9,4 --> 7,6	314.6517
		0.634559141	100	9,7 --> 7,5	445.16684
		0.043971790000000004	4	7,5 --> 5,3 --> 5,5 --> 7,5	1043.3506
		0.022724117000000002	2	8,7 --> 6,5 --> 4,7	1293.5874
		0.021595482000000003	2	4,7 --> 6,7	1699.6761
		0.019070859000000003	2	6,7 --> 4,9	2385.8667
		0.01845737400	2	4,2 --> 3,1	2555.335

		0000002			
		0.016292574	2	3,1 --> 2,1	2747.3652
Negras	3	0.22784642400000002	34	5,2 --> 7,4 --> 9,4 --> 9,2 --> 11,2 --> 9,4 --> 11,6	145.45154
		8.278386282000001	1511	6,6 --> 7,7	190.5181
		0.333693558	34	6,8 --> 8,6 --> 10,6 --> 8,8 --> 10,10	523.55884
		6.128530584	827	7,7 --> 8,6	581.0792
		0.265445433	39	6,7 --> 8,7	776.46954
		5.065776088000001	930	5,8 --> 6,8	936.13715
		0.12889925800000002	24	5,9 --> 7,7 --> 9,7 --> 11,7 --> 13,5	1501.461
		3.1670267	597	13,5 --> 14,4	1723.0101
		3.092796451000003	547	5,6 --> 5,8 --> 7,8 --> 9,6	1929.541
		4.290183088	693	14,4 --> 15,5	2274.1704
		3.953600430000003	797	5,5 --> 7,5	2393.3345
		2.518786063000003	600	8,7 --> 10,5	2554.456
		0.042952466	12	7,5 --> 9,7	2837.7651
		0.974188581	356	5,3 --> 5,5 --> 7,3 --> 5,3 --> 7,5	2953.183
		0.916656201	358	6,8 --> 5,9	3127.8958

		0.003170214	1	5,7 --> 7,9	4152.9805
	3	8.724282765	1714	5,3 --> 7,3	2.4724412
		8.76324971	1792	5,2 --> 7,4 --> 7,2	9.184314
		17.055629098	3096	6,3 --> 8,3	24.382814
		0.040849102000000005	8	7,2 --> 7,4 --> 9,2 --> 11,0	137.3137
		2.741733345	530	5,4 --> 7,4 --> 7,2	158.42223
		10.770461131000001	1797	6,4 --> 7,4	188.97722
		1.6778670660000001	217	6,8 --> 4,6 --> 6,4 --> 8,2 --> 6,2 --> 8,4 --> 8,2 --> 10,0	342.34506
		0.058855381000000005	8	11,0 --> 9,0 --> 7,0	448.40448
		0.055020018000000004	5	8,3 --> 6,3 --> 8,5 --> 10,3	706.77484
		3.327310118	427	5,7 --> 6,8	785.7167
		0.112354619	16	11. 5,9 --> 5,7 --> 7,9 --> 9,9 --> 11,7	1316.7843
		1.934414778	327	12. 7,0 --> 6,1	1267.0746
		0.059347716	10	6,1 --> 8,3 --> 6,3 --> 8,5 --> 10,7 --> 8,9 --> 6,11	2412.3726
		0.188393675	29	10,3 --> 10,5 --> 8,5	3372.9504
		3.271986328	667	7,4 --> 9,6	3638.4783
		3.27374905200	757	6,6 --> 4,6 --> 6,4 --> 8,2 -->	3889.1606

		00003		6,2 --> 8,4 --> 8,6 --> 10,6	
		2.029026182	561	10,6 --> 11,7	4135.5537
		0.0023387770000000002	1	9,6 --> 11,8 --> 9,10 --> 11,10	6126.6084
Blancas	3	4.041183982000001	568	10,4 --> 8,2	13.401844
		5.315540194	680	8,2 --> 7,1	16.462715
		2.8458262050000003	498	10,8 --> 8,6 --> 10,4 --> 8,2 --> 6,0	44.616333
		2.265082903	398	9,6 --> 11,8 --> 9,10	15.284292
		4.504997263	670	9,4 --> 9,2 --> 11,2 --> 9,4 --> 9,6	46.787224
		3.442763137	578	10,9 --> 8,7	-39.725548
		0.03458241	5	10,2 --> 10,4 --> 8,6 --> 8,8 --> 10,8 --> 10,10	95.23398
		0.064695028	7	9,8 --> 7,6 --> 5,8 --> 5,10	366.83868
		3.872164521	509	9,3 --> 9,2	395.88867
		1.6509311910000002	364	5,10 --> 4,11	455.1458
		1.569920793	361	10,7 --> 9,8	476.01257
		0.015770424	3	8,7 --> 10,7 --> 8,9 --> 6,11	747.8159
		1.542608562	368	7,1 --> 7,2	598.043
		0.528539416	122	6,0 --> 8,2	856.9862
		10.7195668080	1868	10,5 --> 8,5	896.53894

		00002			
		5.789939802	1125	8,2 --> 7,2	647.3935
		0.026501915	11	8,5 --> 6,3 --> 4,5 --> 6,7	1529.6414
		0.075669999	41	9,2 --> 7,0	2194.9817
		1.95852107800 00002	695	9,6 --> 8,7	2339.0408
		1.42319383200 00002	514	10,10 --> 9,10	2057.0356
		0.05015659	32	. 9,8 --> 11,10	2963.9177
		0.544482272	302	8,7 --> 7,8	2578.714
		0.03296139500 0000004	26	9,7 --> 7,9	3600.5374
		0.84311491300 00001	572	6,11 --> 6,10	3604.0903
		0.150860336	128	10,6 --> 9,7	3225.5332
		0.013735034	18	6,10 --> 8,8 --> 10,6 --> 8,4 --> 8,2	4414.8145
		0.54059391200 00001	544	9,5 --> 10,6	4507.9277
		0.308478547	225	10,6 --> 8,8 --> 6,10	4802.836
		0.440577205	358	9,7 --> 9,6	4498.164
		0.01537409800 0000001	18	4,11 --> 3,10	4580.097
		0.01656384100 0000003	18	3,10 --> 2,9	4769.343
		0.01398888200	18	2,9 --> 1,8	4959.3794

		0000001			
		0.01549430200 0000001	18	1,8 --> 1,7	5011.666
		0.011216145	18	1,7 --> 0,6	5198.488
		0.010155822	18	6,10 --> 5,9	5215.7876
		0.01212554200 0000001	17	5,9 --> 4,8	5068.838
		0.001008477	10	7,0 --> 6,0	5123,453
	3	3.972784931	568	10,4 --> 8,2	-1.1814876
		0.019934522	3	9,3 --> 11,3 --> 9,1 --> 7,3 --> 5,1	-1.0149
		7.00232456800 0001	1154	10,9 --> 8,7	34.05414
		10.264488416	1607	9,5 --> 9,3 --> 11,3 --> 9,1 --> 7,3	39.03147
		0.02390237800 0000002	4	10,2 --> 10,4 --> 8,4 --> 6,2 --> 4,4	67.04249
		0.05152068500 0000004	7	9,8 --> 7,6 --> 5,4	53.782814
		0.066268806	11	7,3 --> 5,5 --> 7,7	386.3573
		2.713825859	545	10,8 --> 8,6	266.5712
		0.518113233	118	8,7 --> 8,9	524.70337
		3.11888207200 00003	623	8,6 --> 7,6	336.26324
		0.315315917	58	9,4 --> 7,6	689.576
		1.53270466700	382	8,9 --> 8,8	748.0409

		00002			
		3.7045291240000005	1181	9,6 --> 9,8 --> 7,8	754.75165
		0.103141988	37	8,8 --> 6,8 --> 4,6 --> 6,6	1573.4758
		2.097501273	649	10,6 --> 8,8 --> 6,8	1706.5212
		3.6013233870000003	1095	7,6 --> 5,6	1759.2073
		0.7440540720000001	338	10,7 --> 8,7 --> 6,9 --> 6,7 --> 4,5	1908.4463
		0.089409258	49	6,6 --> 4,6 --> 4,4 --> 2,4	2530.4329
		0.448057452	251	5,6 --> 3,4 --> 1,4	2700.4524
		1.3206878080000002	753	1,4 --> 0,5	2936.436
		0.772561037	409	6,8 --> 6,7	2998.7969
		1.2869395240000001	717	22. 2,4 --> 2,3	3031.7905
		2.064814846	1328	7,8 --> 5,6 --> 3,4 --> 1,2	3110.3923
		0.012541452000000002	62	4,5 --> 3,5	3167.8667
Negras	4	13,6984	2589	5,9 --> 7,7 --> 9,9 --> 9,7 --> 11,7 --> 9,9 --> 11,9	134,764
		231,1922	43852	5,3 --> 7,3	223,968
		12,2437	4631	7,4 --> 7,2 --> 9,2 --> 9,4 --> 11,4 --> 9,2 --> 11,2	737,208

		30,4038	26974	6,4 --> 6,2 --> 8,4	960,586
		1,3184	1714	7,3 --> 9,3	1.290,587
		14,7294	28931	5,5 --> 5,3 --> 7,3	1.517,855
		20,3759	32521	7,3 --> 5,3 --> 5,5 --> 7,7	1.729,366
		0,0046	1	7,7 --> 9,9 --> 9,7	2.579,546
Blancas	4	430,6454	94382	10,3 --> 9,2	-6,698
		389,3309	71711	9,4 --> 8,3	1,554
		1,3001	238	9,8 --> 11,6 --> 9,4 --> 7,2 --> 5,2	-27,509
		13,0821	3356	10,2 --> 8,2 --> 8,4 --> 6,4 --> 4,4	60,255
		12,0232	2930	9,3 --> 7,3 --> 5,3	213,154
		1.662,3221	397241	8,3 --> 7,2	274,489
		0,7828	211	5,3 --> 5,5 --> 7,7	452,353
		0,3830	386	9,7 --> 7,7	568,885
		53,7261	53201	10,5 --> 8,7	446,728
		2,3525	2575	9,5 --> 7,7	675,430
		36,9709	44183	10,7 --> 10,5 --> 8,7 --> 6,7	993,275
		0,3438	725	10,9 --> 10,7 --> 8,5 --> 6,7 --> 4,7 --> 6,9 --> 4,9	2.324,794
		0,0059	1	7,2 --> 5,0 --> 5,2	4.014,817

Referencias bibliográficas

Introducción:

http://thedailybytes.wordpress.com/2008/10/08/primer-juego-de-ordenador/

http://www.rae.es

Información Camelot:

http://www.worldcamelotfederation.com

http://en.wikipedia.org/wiki/Camelot_(board_game)

Código de árbol de búsqueda:

http://chessprogramming.wikispaces.com/Alpha-Beta

Tipos de juegos:

Carlos Linares López.”Algoritmos de búsqueda y métodos de aprendizaje estadísticos en el juego de Othello”

Algoritmo alfa-beta:

http://chessprogramming.wikispaces.com/Tony+Marsland#Selected%20Publications-1983

Algoritmos búsqueda recursivos:

Carlos Linares López.”Algoritmos de búsqueda y métodos de aprendizaje estadísticos en el juego de Othello”

Técnicas IA:

Paul S.Rosenbloom, “A World-chapionship-Level Othello program”

Mehdi Samadi, Jonathan Schaeffer, Fatemeh Torabi, Majid Samar, Zohreh Azimifar, “Using Abstraction in two-player games”, 2008.

Introducción a Java:

Java 2. Editorial McGraw-Hill. Jesús Sanchez Allende, Gabriel Huecas Fernandez, Baltasar Fernandez Manjón, Pilar Moreno Díaz

http://www.javaworld.com/

http://www.programacion.com/java

jDialog:

http://download.oracle.com/javase/1.4.2/docs/api/javax/swing/JDialog.html

http://www.chuidiang.com/java/novatos/JFrame_JDialog.php

jShowMessage:

http://www.delphibasics.co.uk/RTL.asp?Name=ShowMessage

http://www.roseindia.net/java/example/java/swing/ShowMessageDialog.shtml

jInternalFrame:

http://download.oracle.com/javase/1.4.2/docs/api/javax/swing/JInternalFrame.html

http://foro.elhacker.net/java/jinternalframe-t328861.0.html

http://www.javabeginner.com/java-swing/java-jinternalframe-class-example

jframe:

http://download.oracle.com/javase/6/docs/api/javax/swing/JFrame.html

http://www.apl.jhu.edu/~hall/java/Swing-Tutorial/Swing-Tutorial-JFrame.html

http://perseo.cs.buap.mx/~danguer/projects/curso_java/manual/node22.html

http://www.forosdelweb.com/f45/formar-correcta-para-jframe-556910/

mouselistener:

http://download.oracle.com/javase/tutorial/uiswing/events/mouselistener.html

http://download.oracle.com/javase/tutorial/uiswing/events/mouselistener.html

http://www.tutorial-lab.com/tutoriales-java/id154-mouselistener-con-java.aspx

http://www.java-tips.org/java-se-tips/java.awt.event/how-to-use-mouse-events-in-swing.html

jToggleButton:

http://download.oracle.com/javase/1.4.2/docs/api/javax/swing/JToggleButton.html

Timer:

http://download.oracle.com/javase/6/docs/api/javax/swing/Timer.html

http://java.sun.com/products/jfc/tsc/articles/timer/

jeditorPane:

http://download.oracle.com/javase/1.4.2/docs/api/javax/swing/JEditorPane.html

jTextPane:

http://www.chuidiang.com/java/ejemplos/JEditorPane-JTextPane/JEditorPane-JTextPane.php

SimpleAttributeSet:

http://download.oracle.com/javase/1.4.2/docs/api/javax/swing/text/SimpleAttributeSet.html

http://www.java2s.com/Tutorial/Java/0240__Swing/UseSimpleAttributeSetwithJTextPane.htm

Estilos:

http://javapiola.blogspot.com/2009/11/tutorial-de-jtextarea-en-java.html

Cronometro, hilos:

http://msdn.microsoft.com/es-es/library/system.windows.forms.application.doevents%28VS.80%29.aspx

http://download.oracle.com/javase/tutorial/uiswing/events/index.html

http://livedocs.adobe.com/flash/9.0_es/ActionScriptLangRefV3/flash/utils/Timer.html

http://es.w3support.net/index.php?db=so&id=391621

Imágenes:

http://www.exampledepot.com/egs/javax.swing.text/tp_ImageText.html

http://www.programacion.com/foros/java-basico/insertar_imagenes_44480

http://foro.chuidiang.com/java-j2se/como-insertar-una-imagen-en-un-jtextpane/?wap2