Sistemas Informáticos
Curso 2003-2004

Sistema de Entrenamiento de Ajedrez para Principiantes

Gonzalo Álvarez Moreno
Helios González Egea
José Romero Pérez

Dirigido por:
Prof. Cristóbal Pareja Flores
Dpto. Departamento de Sistemas Informáticos y Programación

Facultad de Informática
Universidad Complutense de Madrid

Tabla de contenidos

1. Resumen

Este proyecto está destinado para aquellas personas que quieran aprender a jugar al ajedrez, o que teniendo algo de conocimiento, deseen perfeccionarlo.

Se disponen de dos módulos. El primero está pensado para un profesor de ajedrez que quiera "construir" ejercicios que luego resolverán los alumnos en el curso. Estos ejercicios se almacenarán en ficheros de texto, que más tarde podrán ser utilizados por el otro módulo.

La segunda parte de nuestro proyecto consiste en una aplicación que permita al usuario realizar un curso de ajedrez ó jugar una partida contra la máquina. Este sistema ofrece la funcionalidad de crear usuarios, cada uno de los cuales deberán introducir un nombre y una contraseña para poder utilizar esta aplicación. De esta forma, cada uno de estos usuarios tendrá un nivel determinado por el número de ejercicios del curso que haya superado. Por otro lado, se podrá practicar 19 tipos de aperturas y disputar partidas contra el sistema que ofrece un ELO máximo de alrededor de 1.200.

TRADUCCIÓN AL INGLÉS:

This project is addressed to those who aim at learning how to play chess or else having acquired a certain knowledge wish to improve it.

We have two modules. The first one is intended for the chess teacher who wants to build up exercises for his students to solve. These exercises are stored up in text files which later on could be used in the other module.

The second part of our project is an application which allows the user to follow a chess course or play a game of chess against the computer. This system offers the possibility of creating users, who should introduce individually a name and a password to be able to use this part. In this way, each of these users will have a certain level fixed by the number of exercises from the course he has succeeded in doing. On the other hand, you will be able to practise 19 types of openings and play games against the system, which offers a maximum ELO of about 1200.

2. Especificación

Lo primero que se hizo fue tener una serie de reuniones con el director del proyecto para que nos informase de los objetivos del proyecto. En estas reuniones se realizó una primera aproximación al alcance del programa centrándonos en las principales funcionalidades que debería tener el programa. Después de estas primeras reuniones el enunciado del programa quedó de la siguiente forma.

Se pide implementar una aplicación que permita a un usuario que no sepa jugar al ajedrez (o que sepa muy poco) realizar una serie de ejercicios incluidos en un curso de aprendizaje de ajedrez. Estos ejercicios irán destinados a perfeccionar poco a poco el nivel de juego del usuario, así mismo se deberá permitir al usuario disputar una partida de ajedrez acorde con el nivel que haya alcanzado.

Con este enunciado y los encuentros con el profesor, se decidió dividir la aplicación en dos programas. El primero sería el programa profesor, y se encargaría de la creación de los distintos ejercicios del curso. El segundo programa sería el programa del alumno, esta aplicación se encargará de ejecutar los ejercicios creados por el profesor y permitir al alumno que los realice. También debe permitir al usuario jugar una partida de ajedrez con distintos niveles de juego. Estos niveles de juego pueden ser elegidos por el usuario o es el programa el que los asigna automáticamente.

Vamos a empezar explicando la especificación de la aplicación del profesor.

2.1. Generación de ejercicios

La aplicación debe permitir la creación de distintos tipos de ejercicios así como la posibilidad de guardarlos en disco. Podemos decir que tiene una funcionalidad que es la creación de ejercicios. Pasamos a explicar esta funcionalidad.

Creación de ejercicios: La aplicación debe tener una lista de tipos de ejercicios posibles a realizar. Estos tipos de ejercicios son los que se encuentran en el documento del curso de ajedrez. El usuario debe poder elegir qué ejercicio quiere crear, estos tipos de ejercicios están fijados de antemano y se incluirán ejercicios de ejemplo. Una vez decidido el ejercicio a crear el profesor irá incluyendo las piezas que formaran parte del ejercicio de manera gráfica. Una vez incluidas las piezas y dependiendo del tipo de ejercicio se añadirá la solución del mismo. Esta solución puede ser una secuencia de movimientos, en este caso el profesor irá indicando los movimientos uno a uno y de manera gráfica. Otra posibilidad es que la solución sea una casilla o un sólo movimiento, entonces el profesor señalará esa casilla o realizará ese movimiento. Por último el ejercicio puede ser contestar una pregunta, la pregunta es del tipo signo ó verdadero/falso. Aquí el profesor lo que debe hacer es dar la respuesta a esta pregunta. Una vez terminada la creación del ejercicio, el usuario podrá guardarlo en disco. La gestión del nombre del ejercicio y de los distintos ejercicios del curso se realizará de manera automática por la aplicación. Por último, señalar que el programa debe controlar que no se realicen movimientos incorrectos. También debe controlar ciertas restricciones que tienen los distintos ejercicios, por ejemplo, un ejercicio de mate debe tener un rey, etc.

2.2. Alumno

Esta aplicación va a constar de distintas funcionalidades básicas. La primera y más importante es la gestión del curso y de los ejercicios del mismo. Dentro de esta gestión, está incluida la gestión de cada usuario. Los avances de un usuario determinado dentro del curso se guardan automáticamente y estarán disponibles al volver a ejecutar la aplicación. Debe permitir a un alumno comenzar el curso y realizar los distintos ejercicios del curso. También debe permitir a un alumno disputar una partida de ajedrez contra la máquina. Por tanto, digamos que la aplicación va a constar de dos funcionalidades claramente separadas; la primera será la gestión y realización de ejercicios de ajedrez y la segunda la posibilidad de jugar una partida de ajedrez contra la máquina. Pasamos a explicar con más detalle las dos funcionalidades.

Gestión de ejercicios: Aquí también incluimos la gestión automática que realiza la aplicación de los avances de cada usuario. El programa debe permitir a varios usuarios realizar el curso y por supuesto estos usuarios no tienen por qué estar en el mismo nivel del curso. El programa guardará en disco la progresión del usuario dentro del curso. Cada vez que el usuario arranque la aplicación, ésta cargará automáticamente el siguiente ejercicio que le corresponde hacer al usuario. Vamos a explicar ahora la gestión de ejercicios. Una vez iniciada la aplicación, el alumno debe de alguna manera señalar al programa que desea comenzar a realizar algún ejercicio. La máquina ya conoce en qué nivel del curso se encuentra el usuario, por lo que cargará el ejercicio que corresponda a ese usuario. Una vez cargado lo mostrará por pantalla en forma de tablero de ajedrez. Al mismo tiempo deberá mostrar alguna ayuda al alumno para que pueda finalizar el ejercicio correctamente. Una vez hecho esto y dependiendo del tipo de ejercicio que sea, es el alumno el que comenzará a realizar el ejercicio. Esto se hará bien moviendo piezas o bien pinchando encima de algún botón o de alguna casilla del tablero, todo ello dependiendo de la clase de ejercicio. Los movimientos que realice el usuario se deben comprobar que sean correctos. Si el ejercicio requiere algún tipo de movimiento por parte de la máquina, la aplicación realizará ese movimiento. El movimiento a realizar lo habrá cargado del ejercicio. Si el ejercicio es correcto se podrá pasar a realizar el siguiente ejercicio del curso, sino se podrá repetir. Hay ciertos ejercicios que quizá requieran que la máquina “piense”, es decir, que entre en juego la segunda funcionalidad de la aplicación (la posibilidad de jugar una partida de ajedrez); en estos casos en vez de leer el movimiento a realizar desde un archivo (como en los otros ejercicios) se usará el módulo de juego para que la máquina dé una respuesta razonable.

Gestión de partidas de ajedrez: El alumno puede disputar una partida de ajedrez en cualquier momento del curso. La interfaz gráfica dispondrá de alguna manera (botón, menú) de ofrecer esta funcionalidad al alumno. Dentro de esta opción se podrán elegir distintos niveles de dificultad ó que sea la máquina la que ajuste el nivel automáticamente. Una vez seleccionada esta opción se creará un tablero con todas las piezas en sus posiciones iniciales. Por cada movimiento del usuario, el programa realizará otro movimiento. La aplicación comprobará que los movimientos del usuario sean correctos y no permitirá la realización de un movimiento incorrecto. Por supuesto la máquina no realizará movimientos que no permitan las reglas del ajedrez. Esta sucesión de jugadas continuará hasta que uno de los dos jugadores (alumno o máquina) dé jaque mate al otro o, hasta que el usuario decida terminar la partida.

2.3. Análisis de requisitos

La aplicación deberá poder ser ejecutada en un ordenador de sobremesa de gama media. Aunque debido a la posible dificultad de conseguir que juegue de manera aceptable al ajedrez en un tiempo razonable en un ordenador de gama media (*Pentium III*, 128Mb *Ram*, etc), se podrán realizar pruebas en ordenadores de gama alta (*Pentium IV*, 512Mb *Ram*, etc). Por supuesto la realización del curso de ajedrez debe poder realizarse en un ordenador de los llamados de gama media ya que esta es la parte central del proyecto. La interfaz gráfica debe poder ejecutarse en ordenadores que no dispongan de tarjetas gráficas de última generación. Una tarjeta de video normal, 16 Mb, debería bastar.

No hará falta ningún tipo de conexión de los ordenadores ya que el programa se ejecutara de forma local en el ordenador.

Se debe realizar un estudio en profundidad sobre las tecnologías a usar en el proyecto. Pensamos que usar tecnologías conocidas por los integrantes del grupo será mejor que enfrascarse en un estudio de nuevas tecnologías. Aunque este punto todavía no está decidido seguramente será necesario estudiar algoritmos de computación de ajedrez para conseguir que el programa pueda jugar.

2.4. Elección del lenguaje

El lenguaje elegido para la implementación del proyecto a sido C++. El entorno de desarrollo elegido es *Borland C++ Builder.*

2.4.1. Ventajas

Se decidió usar C++ en la realización y desarrollo del proyecto debido a varias razones que vamos a pasar a enumerar. La primera y principal es por eficiencia. Un programa de ajedrez necesita de muchos cómputos por lo que el lenguaje a usar debe ser un lenguaje rápido, que nos permita programar al suficiente bajo nivel para optimizar al máximo el código. Por esto C++ fue la opción que se pensó en un primer momento. Esta eficiencia se pensó tanto en tiempo como en espacio, con C++ podemos controlar la memoria usada en cada instante del programa.

Dentro de las múltiples plataformas para desarrollar bajo C++ se eligió *Borland C++ Builder 5.0*, debido a que todos los integrantes del grupo estábamos familiarizados con el entorno. También se eligió debido a las facilidades que nos proporcionaba (facilidad en la creación del interfaz gráfico, potente editor de código, etc.).

Relacionado con la interfaz gráfica varios integrantes del grupo conocían la librería *OpenGL* y la habían usado bajo este entorno.

2.4.2.Inconvenientes

Poca compatibilidad, es decir, una aplicación *Borland C++ Builder* no es tan fácilmente exportable como una aplicación Java. Esto nos puede llevar a una complicación, si queremos que nuestra aplicación esté disponible en Internet, el trabajo a realizar sería enorme comparado con el que se haría en Java (programar un *applet*) y habría que realizar un estudio de tecnologías que podría ser inabordable.

Dificultad en la depuración de código. *Borland C++ Builder* no tiene una herramienta de *debuger* tan potente como lo puede tener un entorno de desarrollo Java (*JBuilder* por ejemplo).

2.4.3.Conclusiones

Por todo esto, había dos lenguajes que teníamos en mente Java y C++, pero sopesando los pros y los contras y estudiando lo que queríamos conseguir de la aplicación, se decidió usar C++. Esta elección influirá mucho en la realización del proyecto. Con esta decisión estamos centrándonos en aspectos del proyecto más algorítmicos y dejando de lado cosas como la exportabilidad y el desarrollo web del proyecto.

2.5. Curso de ajedrez

En este punto vamos a explicar el contenido del curso de ajedrez que deberá seguir el usuario del proyecto. Vamos a explicar en orden secuencial lo distintos puntos que tendrá el curso y una explicación de cada uno.

2.5.1. Introducción al ajedrez (Nivel inicial)

Aprendizaje del tablero: Pequeña introducción al tablero del ajedrez (aspecto, nomenclatura, etc.)

Movimiento de piezas: Para saber jugar al ajedrez, hay que saber cómo se mueven las piezas. En este punto se estudiarán todas las piezas de las que se compone el ajedrez, los valores de estas piezas y cómo mueven.

- Identificación de piezas: Hay que mostrar al usuario la forma que tienen las distintas piezas y su nombre.
- Objetivos del ajedrez: Se explicará cuál es el objetivo de la partida (capturar al rey) y la importancia de las distintas piezas para lograr el objetivo.
- Valoración de piezas: Se explicará cuáles son las piezas más importantes y por qué.
- Movimiento de piezas: Se mostrará como mueven las distintas piezas.

2.5.2.Reglas básicas del ajedrez (Nivel medio)

Capturar piezas, jaque, jaque mate y ahogado: En este punto se estudiará como comen las distintas piezas del ajedrez. Se explicará también qué es un jaque y qué piezas intervienen en él, qué es un jaque mate y qué significa estar ahogado.

- Captura de piezas: Qué es y cómo se realiza. Se explicará cómo comen las distintas piezas.
- Jaque: Se mostrará lo que es un jaque y para qué sirve.
- Jaque mate: Se explicará lo que es y su importancia. Se mostrará como dar jaque mate en un movimiento con distintas piezas y cuando no se puede dar mate.
- Introducción al ahogado: Se explicará qué es y su significado

Movimientos avanzados (enroque, comer al paso, etc.): Aquí se estudiarán movimientos avanzados del ajedrez que no son necesarios para poder jugar (se puede jugar perfectamente sin ellos) pero que son necesarios para alcanzar un cierto nivel de juego.

- Enroque: Qué es y para qué sirve. Se explicará cuándo se puede hacer y cuando no.
- Comer al paso: Qué es y para qué sirve. Se explicará cuándo se puede hacer y cuando no.
- Coronar: Qué es y para qué sirve. Se explicará cuándo se puede hacer y cuando no.

Defensa del rey: Una vez que se ha aprendido a dar jaque hay que aprender a defenderse de él. En este punto se intentará explicar cuáles son las distintas alternativas que tenemos para defender a nuestro rey.

- Defensa: Se mostrarán las distintas alternativas que se tienen para salir de un jaque y no caer en jaque mate. Se explicará cómo huir con el rey, cómo cubrirlo y cómo comer la pieza atacante.
- Ahogamiento: Se explicará cuándo nos beneficia y cuándo no. Se darán consejos para evitar que se produzca cuando estemos atacando al rey contrario y cómo provocarlo si nos están atacando a nosotros y nos benefician unas tablas.

2.5.3.Nivel avanzado

Juego avanzado:

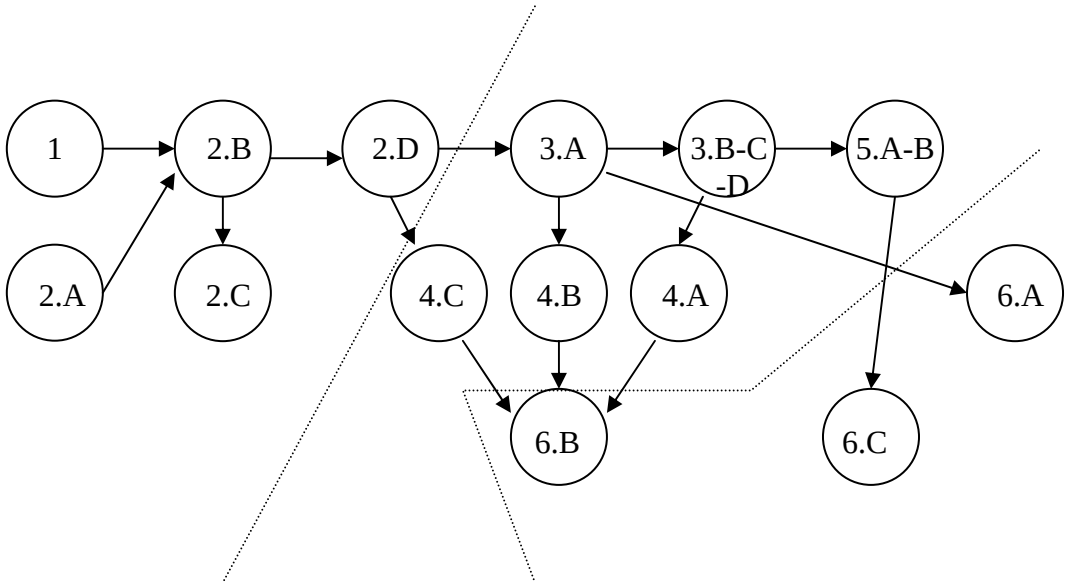
- Finales: Se enseñará cómo acabar una partida con el menor número de movimientos posibles. Se mostrará cómo dar jaque mate al rey con distintas combinaciones de piezas. A su vez se enseñará cómo defenderse en estos casos.
- Juego medio: Distintas tácticas para desarrollar durante el juego. Se enseñará cómo aplicar estas técnicas para conseguir mejorar el nivel de juego. Algunas técnicas son
  - Sacrificio de piezas
  - Rayos X
  - Ataque doble

- Clavadas
- Intercepciones
- Desviaciones
- Aprender a suministrar el tiempo del juego

o Importancia de la posición de las piezas en el tablero

o Aperturas: Qué son y para qué se usan. Se mostrarán ejemplos de aperturas famosas y se explicaran sus ventajas e inconvenientes mediante comentarios que el programa irá mostrando por pantalla para ayuda del usuario. Se intentará explicar la importancia que tiene una buena apertura para el desarrollo de la partida.

Grafo de temas. Se explica en el siguiente grafo qué temas son necesarios para comprender a otros.



Nivel inicial: Introducción al ajedrez, a sus piezas y a los movimientos de éstas. En este nivel se pueden realizar ejercicios sencillos pero no jugar partidas.

Nivel medio: Reglas del ajedrez. Según se vaya avanzando en este nivel se podrán ir jugando partidas de ajedrez más complicadas. Al finalizar el nivel, el usuario estará en disposición de poder jugar una partida de ajedrez.

Nivel avanzado: Técnicas avanzadas del juego (aperturas, medios, finales). Al finalizar el nivel, el usuario podrá disputar partidas contra un experto humano y hacerle frente.

2.6. Ejercicios de ajedrez

2.6.1. Aprendizaje del tablero

Ejercicio de aprendizaje de las posiciones del tablero: Se dan distintas posiciones del tablero y se pide al usuario que las señale. Este ejercicio sirve para iniciar al usuario en la forma del tablero y sus posiciones.

Ejercicio de aprendizaje de nomenclatura de ajedrez: Dado un tablero con un determinado numero de piezas, se va a proporcionar al usuario un movimiento, por ejemplo c4 o Ta6 o Dxd3, y se le pide que reproduzca ese movimiento. Este ejercicio intenta introducir al usuario en la nomenclatura usada en el ajedrez para representa las piezas y sus movimientos.

2.6.2. Aprendizaje de piezas

Ejercicio de aprendizaje de nombres de piezas y su valor: Dadas distintas piezas se pedirá al usuario identificarlas y luego ordenarlas por orden de valor de mayor a menor.

Ejercicios de aprendizaje de movimiento de piezas: Se mostrará un tablero con distintas casillas "obstáculo", que son posiciones por las que una pieza no podrá pasar. Se dará una pieza y una casilla destino y se pedirá al usuario que llegue a esa casilla destino con el menor número de movimientos posibles. Este ejercicio introduce al usuario en el movimiento de las piezas.

2.6.3. Aprendizaje de captura de piezas

Comer una pieza desprotegida: Se dará al usuario una posición del tablero y se le pedirá que coma la pieza del oponente que está desprotegida (sólo habrá una)

Capturar una pieza con el menor número de movimientos posibles: Ejercicio parecido al de movimiento de piezas, lo que pasa es que ahora el objetivo es la captura de una determinada pieza del contrincante. El ejercicio se realizará con todas las piezas.

Capturar la pieza con más valor: Se darán al usuario distintas situaciones en las que se puede comer varias piezas, el usuario deberá elegir comer la pieza de mayor valor.

2.6.4. Aprendizaje de jaque

Dar jaque: Dada una posición del tablero, se pedirá al usuario que con un movimiento dé jaque al rey.

Dar jaque y amenazar otra pieza: Igual que el ejercicio anterior pero ahora además de dar jaque, el usuario debe elegir una posición en la que también amenace a otra pieza del rival.

2.6.5. Aprendizaje de jaque mate

Dar jaque mate: Se pedirá al usuario que con un solo movimiento dé jaque mate.

Dar jaque mate avanzado: Se pedirá al usuario que dé jaque mate con dos movimientos.

2.6.6. Aprendizaje de movimientos avanzados

Enroque: Se darán distintos tableros y se pedirá al usuario decir si es posible realizar el enroque. Los ejercicios se propondrán tanto para el enroque corto como el enroque largo.

Comer al paso: Se darán distintas situaciones y se pedirá al usuario que conteste si se puede o no comer al paso, si es así el usuario deberá realizar el movimiento.

Nota: Lo mismo sería una buena idea no hacer ejercicios concretos de estos movimientos e introducirlos "camuflados" en ejercicios anteriores, de forma, por ejemplo, que para dar mate haya que promocionar a un peón o que para comer una pieza haya que comer al paso. También se podrían realizar ambas opciones.

2.6.7.Defensa básica

Defender al rey: Se dará un tablero en el que están dando jaque al rey del usuario y se le pide que lo defienda y evite que se lo coman. El usuario podrá defenderlo de tres maneras (escapar, comer la pieza que da jaque o tapar el rey)

Ahogamiento: Se produce cuando el rival no puede mover ninguna pieza. Se dan distintos tableros y se pide al usuario decir si el rey está o no ahogado.

2.6.8.Aprendizaje de aperturas

Inicialmente aparece el tablero con todas las piezas de cada jugador y se permitirá al usuario mover piezas de tal forma que dichos movimientos satisfagan las aperturas que están almacenadas en la base de conocimiento.

El jugador practicará con el ordenador los primeros movimientos de una partida. Este ejercicio se acabará o bien cuando se termine la apertura efectuada, o bien cuando el jugador se aparte de la línea establecida (en ocasiones, se le podrá avisar al jugador de la razón por la cual el movimiento efectuado no es el adecuado y debería haber continuado con la línea de la apertura).

Habrá ejercicios para cada tipo de apertura:

Aperturas abiertas: El objetivo estratégico es crear estructuras aptas para el juego vivo y arriesgado, donde desde un principio resultan posibles los golpes de efecto rápido. Este tipo de aperturas se inicia siempre con la jugada 1.e2-e4, aceptando las negras el desafío con la respuesta 1. ...,e7-e5.

Ejemplos: Apertura Ruy López, Apertura Italiana, Gambito del rey, Apertura Escocesa, Apertura Del centro, Apertura Vienesa, Apertura De alfil, Apertura Cuatro caballos, Apertura Ponziani, Defensa Petrov, Defensa Philidor, Defensa Dos caballos, Defensa Hungara, Gambito leton.

Aperturas semiabiertas: En ellas el conductor de las negras declina desde el primer momento del sistema elegido por su rival y trata de conducir la batalla con ritmo más lento. Son aquellas en que a la jugada 1.e2-e4 responden las negras con jugada distinta de 1. ...,e7-e5.

Ejemplos: Defensa Francesa, Defensa Siciliana, Defensa Caro-kann, Defensa Alekhine, Defensa Escandinava, Defensa Pirc, Defensa Robatsch, Defensa Nimzovich, Defensa Owen.

Aperturas cerradas: Se caracterizan por el hecho de que las blancas inician la lucha con cualquier jugada que no sea 1.e2-e4.

Ejemplos: Apertura Catalana, Ataque Torre, Apertura Trompowsky, Apertura reti, Apertura Inglesa, Apertura Bird, Apertura Larsen-Nimzovich, Apertura Sokolsky, Defensa Holandesa, Defensa Nimzoindia, Defensa Bogoljubov, Defensa India de dama, Defensa India de rey, Defensa Grünfeld, Defensa Benoni.

Aperturas que se piensan incluir: Apertura Española, Defensa de los dos Caballos, Apertura Italiana, Apertura Escocesa, Variante de la Apertura Escocesa, Gambito de Rey, Gambito Evans, Defensa Siciliana, Defensa Siciliana de Paulsen, Defensa Siciliana de Paulsen modificada, Defensa Siciliana de Boleslavsky, Defensa Siciliana de Rauzer, Defensa Siciliana de Fischer, Gambito Escocés, Defensa Francesa, Defensa Ufimsev, Defensa Escandinava, Defensa Alekhine, Defensa Caro Kann.

2.6.9. Aprendizaje de gambitos

Un determinado jugador "rompe con la teoría" que aconseja que a cada tiempo corresponda la movilización de una pieza.

Se introducirían ejercicios en los que dentro de las propias aperturas, el ordenador realizaría esta especie de regalos; de esta forma se explicaría al usuario el por qué de estos en principio estúpidos movimientos, desarrollando luego el ordenador movimientos posteriores que justifiquen ese regalo.

2.6.10. Aprendizaje de celadas

No conviene subvalorar el conocimiento de estas trampas. Introduciremos ejercicios que parten de posiciones normales y frecuentes. De esta forma, el usuario estará facultado para apreciar fácilmente el riesgo que estas posiciones normales encierran, previendo lo necesario para evitar sus efectos.

2.6.11. Aprendizaje del juego medio

El tiempo: El ejército que haya movilizado sus efectivos con mayor economía de tiempo, es decir, quien disponga de mayor radio de acción para sus armas, empezará antes la táctica de maniobras y podrá alcanzar la ventaja.

Para apreciar mejor lo expuesto, examinaremos casos prácticos.

El espacio: Se expondrán ejercicios en los que se manifieste la idea de que quien más espacio domine, tendrá mejores posibilidades tácticas.

Sacrificio por desarrollo: La entrega de un peón es un buen negocio cuando a cambio de ello se acelera la entrada en el medio juego. Tendremos ejercicios en los que pueda comprobarse lo expuesto.

Ataque de las minorías: Es el que se realiza por los peones en cualquier flanco. En estos ejercicios comprobaremos la virtualidad de esta acción de infantería.

Combinación: Es una estrategia apoyada en concepciones imaginativas, en virtud de la cual un jugador antevé una serie de golpes de tal manera enlazados entre sí que consiguen la victoria inmediata o bien una ventaja suficiente. Podremos ver este estilo mediante algunas partidas famosas que han tenido lugar.

Cadena de peones: La estructura o formación de peones tiene un singular valor, sobre todo con vistas a la táctica que requiere el tratamiento de los finales. Además estamos ante la única arma incapaz de rectificar su andadura. Permite lo que se llama "cerrar el juego", impidiendo las irrupciones de las baterías adversarias. Elegiremos ejercicios para ver de cerca este fenómeno.

El "*hole*": Es toda casilla que quedó desprotegida a causa de los movimientos de peones. Mostraremos ejercicios de partidas en las que la influencia del "*hole*" quedó bien de manifiesto.

Además, se incluirán los siguientes tipos de ejercicios de jugadas más específicas:

Ejercicio de ataque doble: Se le presentará al jugador un tablero con la posibilidad de realizar un ataque doble y deberá descubrirlo y realizarlo.

Ejercicio de defensa de ataque doble: Se le presentará al usuario una amenaza de ataque doble por parte del rival y tendrá que defenderse de la mejor forma posible.

Ejercicio de clavada: Se le presentará al jugador un tablero con la posibilidad de realizar una clavada y deberá descubrirla y realizarla.

Ejercicio de defensa de clavada: Se le presentará al usuario una amenaza de clavada por parte del rival y tendrá que defenderse de la mejor forma posible.

Ejercicio de ataque descubierto: Se le presentará al jugador un tablero con la posibilidad de realizar un ataque descubierto y deberá descubrirlo y realizarlo.

Ejercicio de defensa de ataque descubierto: Se le presentará al usuario una amenaza de ataque descubierto por parte del rival y tendrá que defenderse de la mejor forma posible.

Ejercicio de intercepción: Se le presentará al jugador un tablero con la posibilidad de realizar un ataque por medio de una intercepción y deberá descubrirlo y realizarlo.

Ejercicio de defensa de intercepción: Se le presentará al usuario una amenaza de intercepción por parte del rival y tendrá que defenderse de la mejor forma posible.

Ejercicio de desviación: Se le presentará al jugador un tablero con la posibilidad de realizar una desviación y deberá descubrirla y realizarla.

Ejercicio de defensa de desviación: Se le presentará al usuario una amenaza de desviación por parte del rival y tendrá que defenderse de la mejor forma posible.

Ejercicio de rayos X: Se le presentará al jugador un tablero con la posibilidad de realizar un ataque de rayos X y deberá descubrirlo y realizarlo.

Ejercicio de defensa de rayos X: Se le presentará al usuario una amenaza de rayos X por parte del rival y tendrá que defenderse de la mejor forma posible.

2.6.12. Aprendizaje del juego final

Mates de rey robado: Habrá ejercicios en los que el rey se encuentre sólo en el tablero contra distintos tipos de ejércitos.

- Dama y torre
- Dos torres
- Dama y rey
- Torre y rey
- Peón y rey (siendo posible el mate)
- Rey y dos alfiles
- Rey, alfil y caballo

3. Estudios sobre algoritmos eficientes de juego

Este documento muestra el estudio y la investigación que se llevó a cabo para poder hacer que el programa jugase al ajedrez. El documento se divide en tres campos de búsqueda: tablero, árbol de búsqueda y función de evaluación. Vamos a ver cada uno por separado.

3.1. Función de evaluación

El objetivo de este documento es dar unos criterios a tener en cuenta para implementar la función de evaluación del programa de ajedrez. Básicamente esta función consta de dos partes:

- **Evaluación material**: A través de la cual cada una de las piezas tiene un valor predeterminado dado por su potencia de movimiento. Es la característica de cada pieza que más peso debe tener.
- **Evaluación posicional:** Valor añadido a cada pieza según el contexto del tablero, el cuál viene dado por la posición de cada una de las demás piezas que hay en él.

El resultado de esta función de evaluación es la diferencia entre la puntuación dada a las piezas de cada jugador según de qué jugador sea el turno. Esas puntuaciones vienen dadas por la suma de los puntos adquiridos por cada una de las piezas. De este modo, cada pieza obtiene un número de puntos calculado al sumar las bonificaciones ó penalizaciones impuestas por las reglas de dicha pieza.

3.1.1.Torre

Valor material: 500.

Control de casillas: Número de casillas atacadas por una torre incluyendo casillas ocupadas por el enemigo ó por piezas amigas. Cada casilla se puntúa por 1.6.

Cercanía a rey: El factor –1.6 se multiplica por el mínimo de las distancias vertical y horizontal entre la torre y el rey. La distancia vertical (horizontal) es la diferencia absoluta entre el número de columnas (ó filas) de la torre y el rey. Esta característica es una penalización por distancia en vez de un bonus por cercanía.

Torres dobladas: Consiste en una torre que está en la misma fila ó columna que la que se encuentra la torre amiga, sin que intervengan piezas. Debido a que ambas torres se cuentan por separado, un par de torres "dobles" se acredita por 16.

Columnas abiertas: Una columna abierta es aquella que no contiene peones; y una columna semiabierta es aquella en la que no hay peones amigos pero al menos hay un peón enemigo que no está defendido por otros peones enemigos y el cual no puede mover hacia delante una casilla sin ser atacado por uno ó más peones. Se suma 8 por cada columna abierta y 3 por cada columna semiabierta.

Campo enemigo: El bonus por una torre en la séptima columna es de 22.

3.1.2. Caballo

Valor material: 325.

Cercanía del centro: Esta medida se computa como (6 – 2*SUM)*1.6, donde SUM es la suma de las distancias de filas y columnas desde el centro del tablero a la casilla sobre la que está el caballo.

Cercanía a rey: Esta medida se computa como (5 – SUM)*1.2, donde SUM es la suma de las distancias de filas y columnas entre el caballo y el rey enemigo.

Posición inicial: Penalizar con –9.4 si el caballo está en la casilla inicial.

Control de esquinas: Penalizar caballos demasiado cerca de los límites del tablero: -20 si está en una de las 4 esquinas, -15 si está en un lateral sin ser esquina, -5 si esta a una casilla de estar en un lateral.

3.1.3. Alfil

Valor material: 350.

Control de casillas: Esta medida se computa como (SUM-7)*2.4, donde SUM es el número de casillas que el alfil directamente ataca (ó defiende), sin contar el número de casillas ocupadas por peones amigos.

Posición inicial: Penalizar con –11 si el alfil está en la casilla inicial.

3.1.4.Dama

Valor material: 900.

Control de casillas: Sólo se cuentan las casillas no atacadas por piezas enemigas ó por peones. Cada casilla vale 0.8.

Cercanía a rey: Las damas son penalizadas en proporción al mínimo de las distancias horizontal y vertical con respecto al rey enemigo. El número de filas ó columnas se multiplica por –0.8.

3.1.5.Rey

Valor material: Infinito (no infinito literal sino un el mayor valor posible que permita la computadora).

Seguridad del rey: Es el producto de dos factores: f1*f2, donde f1 es la medida de cómo de importante es para un rey ser protegido en esta fase de juego, y f2 es la medida de cómo de bien protegido está el rey. F1 se computa como (NP + QB – 2), donde NP es el número de piezas enemigas que no sean peones y QB es 2 si el enemigo tiene una reina, sino 0. Si esa medida devuelve 0 ó menos, esta propiedad de seguridad del rey se ignora. En la posición de apertura, la importancia es 8; si desaparecen ambas reinas, lo reduce a 5. El efecto de f1 es animar al jugador cuyo rey está expuesto al intercambio de piezas, y a animar a su oponente a evitar estos intercambios.

Por otro lado, con f2 hacemos mención a que el rey es penalizado si no hay por lo menos 2 piezas amigas ó peones en casillas adyacentes (haciendo el rey un movimiento) a él. Si sólo hay una pieza, la penalización es –1.6, y si no hay ninguna, de –4.1. Si no hay peones amigos en al menos una de las columnas adyacentes al rey, el rey es penalizado con –3.6.

Si al menos dos casillas adyacentes al rey son atacadas por piezas enemigas, se añade la penalización de 1.2*(NAT-1), donde NAT es el número de esas casillas atacadas.

Cercanía a centro: Cuando se ha perdido una cantidad suficiente de material por parte de los 2 bandos, el rey tiende a dejar de esconderse y vagar hacia el centro del tablero poniéndose por delante de cualquiera de los peones del tablero. Esta situación sucede cuando el valor material del enemigo baja a 1500 (p.e. una torre, alfil, caballo y 3 peones). Cuando el material es esa cantidad ó una menor, el rey es penalizado por la distancia desde el centro. La penalización es -2*(SUM), donde SUM es la suma de las distancias de fila y columna desde el rey al centro del tablero.

Fin de juego: El rey es penalizado por extraviarse de cualquiera de los peones que quedan. La penalización es de –7.8*(ATPD – 6), donde ATPD es la media de las distancias "*taxicab*" desde el rey a cada peón amigo ó enemigo. La distancia "*taxicab*" es la suma de las distancias de las filas y columnas, como si el rey pudiera mover sólo horizontal ó verticalmente para alcanzar el peón.

Este apartado fue inspirado al observar que varios programas de ajedrez llegaban al final con reyes y cubiertos por peones y entonces acababa la partida en empate debido a que los reyes se movían repetidamente avanzando y retrocediendo con el mismo movimiento.

3.1.6.Peón

Valor material: 100.

Peones doblados: Son 2 ó más peones del mismo color en la misma columna. Se penaliza con –8 por cada peón.

Peones aislados: Son aquellos que no tienen peones amigos en cada una de las columnas adyacentes (ó única columna para los peones de torre). Se penaliza con –20 por cada peón.

Peones "libres": Son aquellos cuyo camino hasta la octava fila no está impedido por bloques ó ataques de peones enemigos. Se bonifica multiplicando el cuadrado del número de filas por un factor cuyo valor básico es 2.3, pero que es modificado por el estado de la casilla en frente del peón. Si la casilla está bloqueada por una pieza enemiga, el factor se reduce en 0.7. Si es atacada por una pieza enemiga, se reduce en 0.4, y si es defendida por una pieza amiga, se incrementa en 0.3.

Peones muy atrasados: Son peones que no están defendidos por otros peones, no pueden ser defendidos si avanzamos otros peones, y si avanza una fila dicho peón, inmediatamente será amenazado por ataques de peones enemigos en ambas columnas adyacentes sin ninguna compensación por la defensa de peones amigos. Se penaliza con –8 por cada peón.

Peones atrasados: Son como los peones muy atrasados excepto que si avanzan una fila, el número de ataques por parte de los peones enemigos excederá en 1 el número de defensas por parte de los peones amigos, en vez de en 2. Se penaliza con –4 por cada peón.

Posición inicial: Se penalizará que los peones de rey y de dama no se hayan movido todavía de su posición inicial con un valor de –8 por cada peón.

Bonificación de peón: Todos los peones en general reciben bonificaciones adicionales dependiendo de la columna en la que se encuentran: se multiplica el número de columna menos 2 por uno de estos factores (0,0,3.9,5.4,7.0,2.3,0,0) correspondientes a las columnas de los peones recorridas de izquierda a derecha. Fíjese que sólo las 4 columnas centrales tienen valores distintos de cero.

3.1.7.Pieza amenazada

Consideramos que una pieza está amenazada si una pieza que no es peón es atacada por una pieza de menor valor, ó si es atacada por una pieza del mismo tipo, ó de mayor valor y no es defendida. Penalizamos con –16 por cada pieza amenazada. Para el caso de encontrar el caso de una doble amenaza por parte de una pieza, se determinará como puntuación posicional del tablero la peor puntuación que se haya encontrado en el árbol hasta ahora.

3.1.8.Función de evaluación Mop-up

Esta función de evaluación se asigna a posiciones en las que un jugador tiene una ventaja material de al menos 400 puntos (4 peones) y su adversario no tiene más de 700 puntos de materia. Además el jugador que está ganando debe tener al menos una reina ó torre, ó no debe tener peones y al menos tener menos piezas con valor de material.

Todas las posiciones del árbol que no satisfagan este criterio son asignadas a la función de evaluación regular anterior. El criterio de no-peones se basa en la noción de que es más fácil promocionar peones que dar jaque mate con menos número de piezas sólo.

Esta función de evaluación consiste en dos fases:

Fase de tablas: comprueba si hay tablas sólo en el caso de que haya un rey en algún bando.

Fase de jaque mate: es la suma de varios términos:

1.- El rey que está amenazado tiene un gran deseo de evitar las esquinas y *corners* del tablero. Luego se le penaliza con –4.7*CD, donde CD es dos veces la suma de las distancias de filas y columnas desde el rey al centro del tablero.

2.- El rey y sus caballos deben estar cerca del rey enemigo para restringir el movimiento del rey enemigo. Entonces se añade la siguiente cantidad para animar a que se alcance dicha situación: por cada rey y caballo sumamos (14 – SUM)*1.6, donde SUM es la suma de las distancias de columnas y filas desde el rey ó caballo al rey enemigo.

3.2. Árbol de búsqueda

3.2.1. Introducción

La manera en que un programa de ajedrez es capaz de jugar y vencer a un contrincante humano no es una cosa misteriosa de inteligencia artificial. Lo que hace un programa de ajedrez es una búsqueda en el espacio de estados. ¿Qué es esto? Como su nombre indica, consiste en buscar un estado; por estado entendemos una disposición del tablero a la que podemos llegar realizando algún movimiento con nuestras piezas. Nosotros lo que intentamos es buscar el estado que más beneficie a nuestro juego, es decir que nos lleve a ganar. Podemos ver el espacio de estados como un árbol de tableros, es en este árbol donde vamos a realizar la búsqueda. Dependiendo de cómo sea esta búsqueda vamos a tener distintos algoritmos, más o menos eficientes, que pasamos ahora a explicar.

3.2.2. Algoritmo *minimax*

La idea de este algoritmo es que existen 2 jugadores y una función de evaluación de una posición determinada del tablero. Un jugador va intentar maximizar esta función de evaluación, es decir, realizar su mejor movimiento posible, y el otro va a intentar minimizarla. Con esta idea en mente se genera un árbol de búsqueda (suponemos que empezamos maximizando) donde los movimientos del jugador que maximiza se llaman nodos maximizadores y los del jugador que minimiza nodos minimizadores. La idea es que el jugador MAX va a realizar todos los movimientos posibles, después el jugador MIN realizara todos sus movimientos posibles de los movimientos generados anteriormente y así sucesivamente. Esto se puede ver como un árbol. La profundidad de este árbol se elige anteriormente y es fija. En este árbol un nodo minimizador va a devolver a su padre el mejor hijo, en este caso el de menor valor de la función de evaluación y el nodo maximizador va a devolver el movimiento que maximice esa función de evaluación. Así empezando desde las hojas y subiendo en el árbol acabaremos eligiendo la mejor jugada posible a realizar.

VENTAJAS:

Es un algoritmo muy sencillo de implementar y que con una buena función de evaluación siempre va a devolver el mejor movimiento a realizar.

INCONVENIENTES:

Para juegos con pocos movimientos es una solución factible, pero para juegos del ajedrez donde podemos encontrarnos nodos donde se pueden realizar del orden de 50 posibles movimientos, el árbol es de tal tamaño que esta solución es inabordable.

3.2.3.Poda alfa beta

Es el algoritmo más usado en los programas de ajedrez; se trata de un algoritmo *minimax* con ciertas modificaciones. La idea de la poda alfa beta es que no podemos explorar todo el árbol de juego debido a su gran tamaño. Para evitar esto, hay ciertas ramas del árbol que van a ser podadas. Esto quiere decir que no vamos a seguir explorando esa rama del árbol, sino que vamos a continuar explorando la siguiente rama. ¿Cómo conseguimos esto? La idea es llevar 2 valores alfa y beta que van a ir siendo modificados a lo largo del algoritmo; podemos ver estos 2 valores como 2 cotas, máxima y mínima de la función de evaluación del tablero de juego. Vamos a explicar el mecanismo. Nosotros tenemos 2 tipos de nodos, los que maximizan (la máquina) y los que minimizan (el usuario). Los nodos que maximizan van a elegir al hijo que mayor puntuación haya obtenido con la función de evaluación (mejor movimiento de la máquina), mientras que los que minimizan eligen el hijo de menor valor (mejor movimiento del usuario). Visto esto, vamos a explicar qué son la alfa y la beta. Alfa, si estamos en un nodo de maximización, es el mayor de los valores de los hijos, y si estamos en un nodo de minimización es el valor del predecesor; alfa es inicializada a menos infinito. Beta, si estamos en un nodo de minimización, es el menor valor encontrado en los hijos, y si es un nodo de maximización es el valor de su predecesor; beta es inicializada a mas infinito. Visto esto, pasamos a explicar cómo funciona la poda alfa beta. Se puede podar por debajo de cualquier nodo de minimización cuyo valor beta sea menor que el valor alfa del predecesor. Se puede podar la búsqueda por debajo de cualquier nodo de maximización cuyo valor de alfa sea mayor o igual que el valor de beta del predecesor. El algoritmo *minimax* quedaría parecido a esto:

- Si el nivel es el superior, a=-infinito, b=infinito
    - Si se ha alcanzado la profundidad máxima, calcular el valor estático de la posición actual; devolver el valor.
    - Si el nivel es de maximización:
- Hasta que se examinen todos los hijos con Alfabeta o hasta a>=b:
- Usar Alfabeta, con los valores actuales de a y b en un hijo, guardar el valor.
        - Si el valor es mayor que a, asignar el nuevo valor a a.
    - Devolver el valor de a.
    - Si no, el nivel es de minimización:
- Hasta que se examinen todos los hijos con Alfabeta o hasta a>=b:
- Usar Alfabeta, con los valores actuales de a y b en un hijo, guardar el valor.
        - Si el valor es menor que b, asignar el nuevo valor a b.
    - Devolver el valor de b.

VENTAJAS:

El algoritmo consigue reducir bastante el espacio de búsqueda con la consiguiente mejora en la eficiencia del programa. Es un algoritmo relativamente sencillo de implementar y aparte es un algoritmo que ha sido muy usado debido a su eficiencia.

INCONVENIENTES:

El algoritmo puede resultar útil para juegos que tienen un espacio de búsqueda relativamente grande, pero en nuestro caso, el espacio de búsqueda es tan elevado que necesitamos otras estrategias para podar más el árbol de búsqueda. Aparte, este algoritmo depende mucho del orden en que se exploren los hijos; si en algún momento la mejor solución a explorar es visitada en último lugar, debemos visitar todos los hijos, con lo que la poda no se produce. Se transformaría el algoritmo en un *minimax*. Esto nos lleva a pensar que quizá necesitemos un orden de búsqueda de las posibles jugadas, pero esto se tratará más tarde.

Los valores de alfa y de beta son muy importantes en este algoritmo; si conseguimos soluciones que consigan acotar estos valores rápidamente, el algoritmo da resultados muy buenos, pero si tenemos mala suerte como en el ejemplo anterior, la poda no es tan buena como nosotros desearíamos.

3.2.4. Matriz de transposición

Se trata básicamente de añadir una tabla hash al algoritmo de ramificación y poda. En esta tabla guardamos la historia de los tableros que hemos visitado anteriormente, la jugada elegida para esa situación y el valor del nodo. Si en un momento de la búsqueda encontramos un tablero que está en la tabla hash, lo único que debemos hacer es devolver el valor calculado anteriormente y no explorar esa rama. En algunas situaciones no es posible usar el valor, pero aplicar a ese nuevo tablero la jugada que utilizamos anteriormente puede hacer que dentro de esa rama la poda sea mayor.

VENTAJAS:

Estudios de esta técnica han demostrado que puede reducir hasta en un factor de 4 el tamaño del árbol de búsqueda. No es una solución compleja algorítmicamente.

INCONVENIENTES:

La tabla hash puede ocupar mucho espacio en memoria, sobre todo si pensamos que cuantos más tableros guardemos en ella más ramas llegaremos a podar. Hay que buscar un compromiso entre el tamaño de la tabla y el número de podas que queremos obtener.

3.2.5.Profundización iterativa

Consiste en ir haciendo llamadas al algoritmo de búsqueda que tengamos pero cada vez con una profundidad mayor, es decir, la primera vez exploramos hasta el nivel 1, después hacemos una búsqueda hasta el nivel 2, después hasta el 4 y así sucesivamente hasta que se exceda un tiempo máximo o una profundidad máxima. La idea es que para la siguiente búsqueda nosotros vamos a usar los resultados de la búsqueda anterior. Por ejemplo nosotros vamos a llegar a una profundidad de 6 y ya hemos investigado hasta el nivel 5, por lo tanto ya conozco el mejor movimiento del nivel 5 y las puntuaciones por lo que puedo empezar a explorar por ese movimiento.

El código seria el siguiente:

```
for (profundidad = 1;; profundidad++) {
   val = AlphaBeta(profundidad, -INFINITY, INFINITY);
   if (tiempoLimite())
      break;
}
```

3.2.6.Búsqueda por aspiración

Es esencialmente una poda alfa beta con ligeras modificaciones en los valores de alfa y de beta. Inicialmente los valores de alfa y de beta en vez de inicializarse a menos infinito y más infinito respectivamente, se inicializan a valor - desviación y valor + desviación. Donde valor es una estimación del resultado esperado y desviación es una estimación de lo que se va a desviar ese valor.

La técnica más común para implementar esto es usar una profundización iterativa donde los valores de alfa y beta los obtenemos de la iteración anterior, de esta manera conseguimos mejorar el algoritmo de profundización iterativa. El código quedaría como sigue:

```
#define valWINDOW 50
alpha = -INFINITY;
beta = INFINITY;
for (profundidad = 1;;) {
   val = AlphaBeta(profundidad, alpha, beta);
   if (tiempoLimite())
      break;
   if ((val <= alpha) || (val >= beta)) {
      alpha = -INFINITY;    // Nos hemos salido de la ventana
      beta = INFINITY;      //por lo que hacemos otra búsqueda
      continue;             //con la ventana completa
   }
   alpha = val - valWINDOW;  // Ventana para la sig iteración
   beta = val + valWINDOW;
   profundidad++;
}
```

VENTAJAS:

Conseguimos acotar mucho más la búsqueda que en el algoritmo anterior.

INCONVENIENTES:

La dificultad de encontrar una estimación "buena" del valor que nos va a devolver un nodo determinado. Otro inconveniente son los resultados que salgan fuera del valor de alfa y beta que hemos estimado, lo que produciría una nueva búsqueda en ese nodo. Juntándolo con la profundización iterativa, el primer problema queda en parte solventado.

3.2.7.Búsqueda de la variación principal

Este algoritmo se basa en la ordenación de los nodos a explorar, esta idea de la ordenación puede ser incluida en otros algoritmos, por esto ampliaremos esta idea con más detalle después. El algoritmo lo que intenta es hacer una búsqueda en el primer nodo que, si está bien ordenado, dará posiblemente el mejor valor de la función de evaluación posible. Con esta idea vamos a investigar el resto de los nodos con el valor de alfa igual al valor del nodo que ya hemos explorado y el valor de beta como alfa más uno. Con este valor de alfa y beta, y si hemos calculado bien la mejor solución a explorar, la poda del resto de los nodos va a ser inmediata. Si la ordenación de los nodos a explorar no fuese perfecta y hay algún nodo cuyo valor se salga del rango alfa-beta, una nueva búsqueda debe ser hecha. La idea del algoritmo es que las veces que funcione se gana mucho más tiempo que las veces en las que hay que volver a buscar. El código del algoritmo sería el siguiente:

```
int AlphaBeta(int depth, int alpha, int beta)
{
                BOOL fFoundPv = FALSE;

                if (depth == 0)
                        return Evaluación();
                GenerarMovimientos();
                while (hayMovimientos()) {
                        hacerSigMovimiento();
                        if (fFoundPv) {
                        val = -AlphaBeta(depth - 1, -alpha - 1, -alpha);
                                if ((val > alpha) && (val < beta)) //
Comprobar si hay fallo
                                val = -AlphaBeta(depth - 1, -beta, -alpha);
                                } else
                                        val = -AlphaBeta(depth - 1, -beta,
-alpha);
                        deshacerMovimiento();
                        if (val >= beta)
                                        return beta;
                        if (val > alpha) {
                                        alpha = val;
                                fFoundPv = TRUE;
                                }
                }
                return alpha;
}
```

VENTAJAS:

Algoritmo que si funciona correctamente reduce en una cantidad bastante notable el tiempo de computación. No es un algoritmo complejo computacionálmente.

INCONVENIENTES:

Dificultad de la ordenación, no es fácil encontrar criterios para ordenar de mejor a peor un tablero. Se podría usar la función de evaluación pero entonces la dificultad estribaría en encontrar una función de evaluación suficientemente buena para que los posibles errores sean el menor número posible.

3.2.8.Test de la memoria creciente

Aquí dentro se encuentra toda una familia de algoritmos que se basan en usar un tamaño de ventana, diferencia entre beta y alfa, lo menor posible, es decir, beta = alfa-1. Se llama al algoritmo a una profundidad determinada un cierto número de veces. Cada vez que llamamos variamos el valor de alfa. Al tener un tamaño de ventana tan pequeño nos aseguramos que se van a explorar pocas ramas del árbol. Debemos hacer la exploración en el árbol con distintas alfa para no saltarnos soluciones buenas que se podrían no explorar por una mala elección del alfa inicial. Es muy importante en este algoritmo disponer de una tabla de transposición de un tamaño considerable ya que va a haber muchos nodos (tableros) en los que se repita la búsqueda.

VENTAJAS:

Exactamente igual que el algoritmo anterior, si funciona bien, la reducción en el tiempo de cómputo es considerable.

INCONVENIENTES:

Es muy importante la elección del primer valor de alfa y la variación que se le va a aplicar a este valor en las siguientes llamadas al algoritmo. Es un método que con una mala elección de alfa y beta puede llegar a tardar más que una simple poda alfa beta. La tabla de transposición a usar debe ser considerablemente grande con el consiguiente gasto en memoria.

3.2.9.Poda de transposición creciente

Se trata de, antes de evaluar los hijos de un nodo, comprobar que ninguno de estos hijos va a ocasionar una poda porque se encuentra en la tabla de transposición. Si se encuentra, entonces ya no haría falta explorar más en esa rama.

VENTAJAS:

Método que puede reducir hasta en un 20% el tamaño del árbol. Es un método relativamente fácil de programar.

INCONVENIENTES:

Hace falta tener una tabla hash bien diseñada y que ocasione pocos falsos aciertos. Esto quiere decir que a 2 tableros le asignemos el mismo identificador cuando en realidad son distintos. Si esto ocurre, el valor que devolvamos al árbol es un valor incorrecto de la evaluación de ese tablero.

3.2.10. Heurística "*killer*"

Se trata de tener varias jugadas "matadoras" en cada nivel de profundidad. La idea es que si un movimiento es muy bueno en un tablero, también lo será en tableros de la misma profundidad. Una jugada matadora puede ser la que ocasione un corte en un nivel de profundidad. Pero no sólo son este tipo de jugadas, también pueden ser las que ocasionen un jaque, las que coman a la dama rival, etc. Estas jugadas serán las primeras en ser evaluadas en los nodos de ese nivel de profundidad.

VENTAJAS:

Es una solución muy sencilla y que aumenta la probabilidad de podar alguna rama del árbol. Se podría extender la idea de jugada matadora no sólo a un nivel de profundidad sino a todo el árbol.

INCONVENIENTES:

Hay que decidir qué jugadas son matadoras y qué jugadas no lo son.

3.2.11. Heurística de historia

Usa la idea presentada anteriormente de guardar jugadas matadoras para ser aplicadas en todo el árbol. Lo que pasa es que esta solución lo lleva más lejos y posee una tabla con estadísticas de las mejores jugadas para poder ser aplicadas en determinadas situaciones del juego.

VENTAJAS:

Tenemos en cuenta la historia de esta partida para realizar buenos movimientos. Es posible "aprender" los defectos del contrincante y sus puntos débiles.

INCONVENIENTES:

Dificultad para crear la tabla de estadísticas. Hay qué elegir qué valores estadísticos son interesantes para ser guardados y cuáles no. La complejidad de esta solución hace que no sea interesante.

3.2.12. Heurística de movimiento nulo

La idea es que cuando estemos en una situación del tablero muy buena, no hacemos en nuestro turno ningún movimiento y pasamos al turno del adversario. Del adversario vamos a jugar hasta una determinada profundidad, esta profundidad es menor que la profundidad máxima del árbol. Se basa en la idea de que el peor movimiento a realizar es no realizar ningún movimiento. Si el usuario realiza un movimiento y nuestra situación sigue siendo muy buena podemos decidir podar ahí sin tener que haber explorado todos nuestros posibles movimientos. Otra situación es que si tenemos posibilidad de comer una pieza, nosotros pasamos el turno al contrario para que mueva. Entonces nos damos cuenta que la situación es mejor que antes, esto significa que si hubiésemos comido nos habrían comido a nosotros. Esta situación se da en niveles cercanos al máximo nivel de profundidad. Vamos ahora a ver el código:

```
#define R   2

int AlphaBeta(int profundidad, int alpha, int beta)
{
if (profundidad == 0)
                        return Evaluacion();
    hacerMovimientoNulo();//Cambiar el turno
    val = -AlphaBeta(profundidad - 1 - R, -beta, -beta + 1);
    deshacerMovimientoNulo();
    if (val >= beta)
                        return beta;//Si el contrincante no puede hacer ningún
//buen movimiento
            generarMovimientos();
            while (hayMovimientos()) {
                        hacerSigMovimiento();
                        val = -AlphaBeta(profundidad - 1, -beta, -
alpha);
                        deshacerMovimiento();
                        if (val >= beta)
```

```
                                return beta;
                                if (val > alpha)
                                alpha = val;
            }
            return alpha;
}
```

VENTAJAS:

Bien aplicado este método reduce mucho la búsqueda. Es un método muy sencillo de programar.

INCONVENIENTES:

Puede no detectar jugadas buenas (o malas) que se encuentran a un nivel de profundidad elevado. Hay determinadas jugadas, llamadas *zugzwangs*, en las que el principio en que se basa este método de no realizar un movimiento es siempre peor que realizar alguno, no se cumple. Normalmente todas estas jugadas se dan en los finales de partida. Si nos encontramos con alguna de estas jugadas podríamos obtener resultados erróneos, el contrincante puede darnos mate en 2 o 3 y nosotros no darnos cuenta. De todas formas estas jugadas se dan en raras ocasiones y casi siempre en los finales de partida.

3.2.13. Búsqueda de quietud

La idea es que si llegamos al nivel máximo de profundidad de exploración y la solución parece buena, esto no tiene por qué ser cierto. Puede ser que el siguiente movimiento del contrincante sea muy malo para nosotros. Para evitar esto, sólo se devuelve la evaluación de un tablero cuando éste se encuentre en una situación estable (no hay jaques directos, ni piezas amenazadas, etc). Si no se da este caso, profundizamos unos pocos niveles más pero con una búsqueda más restringida. Es decir, sólo buscamos los movimientos en los que estén implicados alguna captura de piezas, un jaque, promocionar un peón, etc. Para mejorar tiempo en vez de usar la función de evaluación completa en estas extensiones, podemos usar una función de evaluación más reducida. El código del programa sería igual al código de alpha beta pero en vez de llamar a la función de evaluación invocamos a la siguiente función:

```
int Quies(int alpha, int beta)
{
        val = Evaluacion();
        if (val >= beta)
                return beta;
        if (val > alpha)
                alpha = val;
        GenerarMovimientosConCapturas();
        while (hayMovimientos()) {
                hacerSigMovimiento();
                val = -Quies(-beta, -alpha);
                deshacerMovimiento();
                if (val >= beta)
                return beta;
                if (val > alpha)
                alpha = val;
        }
```

```
        return alpha;
}
```

Una de las principales dificultades de este método es la elección de las jugadas a comprobar y el nivel de profundidad al que llegamos explorando. Hay que pensar que el número de jugadas debe ser muy pequeño ya que no podemos perder mucho tiempo en esta búsqueda. Se suele aplicar en estos casos la técnica anterior (movimiento nulo) para descubrir las situaciones malas para nosotros.

Una idea del tipo de jugadas que podemos estudiar es el siguiente:

- Jugadas de jaque.
- La máquina está en jaque y sólo hay un movimiento posible a realizar por ella.
- Cuando hay una secuencia de capturas.
- Extender siempre el mejor movimiento encontrado, esta táctica se usaba en *Deep Thought* aunque ahora pocos programas la implementan.

VENTAJAS:

Evita que el algoritmo dé por buena una jugada cuando en realidad no lo es.

INCONVENIENTES:

Añade tiempo a la búsqueda. Hay que elegir qué tableros son propicios para realizar una profundización mayor y cuáles no.

3.2.14. Ordenación de los movimientos a explorar

Esta idea puede ser aplicada a cualquiera de los algoritmos anteriores. Este método propone un ordenamiento de la lista de movimientos de un nodo del árbol. Es decir, que en un nodo determinado del árbol nos interesaría empezar a buscar por el mejor movimiento posible ya que de este modo no necesitaremos explorar otras ramas.

La dificultad del método consiste en cómo asignar este orden. Un ordenamiento posible es:

- Jugadas de la tabla hash
- Capturas que no parecen perder material
- Jugadas "*killer*"
- Otras jugadas que no capturan
- Capturas en las que perdemos piezas

Por supuesto éste no es el único orden, hay otros ordenes que dividen las jugadas *killer* en *killer* mate y *killer*, además de dividir otros grupos.

Vamos a explicar algunos de estos grupos. En el caso de que haya que comprobar si vamos o no a perder la pieza que movemos. ¿Cómo podemos saber esto? Lógicamente todavía no hemos bajado en el árbol y no sabemos si esa pieza que movemos va o no a ser comida. Necesitaremos una función que nos diga si esa casilla está amenazada o no. Esta función se conoce como SEE (*Static Echange Evaluator*). La SEE no puede ser exacta, debe ser una aproximación lo suficientemente buena al resultado real. Un posible orden dentro de estos grupos es basándose en la puntuación de esta función aunque hay otro orden que funciona bastante bien llamado MVV/LVA (*Most Valuable Victim/Less Valuable Attacker*). Este orden se basa en ordenar los movimientos dependiendo de la pieza que captura y la pieza capturada. El orden de la pieza que captura es peón, caballo, alfil, torre y dama. Es decir, de menor valor a mayor. El orden de la pieza capturada es justamente el inverso: dama, torre, alfil, caballo y peón.

Jugadas "*killer*" son esas jugadas que han provocado un corte en alguna rama de la misma profundidad. Se puede extender a jugadas a cualquier profundidad.

Para el orden interno de "otras jugadas que no capturan" se puede elegir cualquier orden razonable, como valor de la función de evaluación en esos tableros, una función de evaluación reducida (sólo posicional), etc. Siempre hay que tener en cuenta que la ordenación gasta tiempo y que debemos encontrar un compromiso entre el tiempo gastado en la ordenación y la perfección de esta ordenación.

3.2.15. Mate

En algunos juegos de ajedrez la detección de si un en un tablero hay jaque mate o no, se encuentra en la función de evaluación. Sin embargo, esta comprobación se puede encontrar en la búsqueda. ¿Cómo?, es muy fácil. Habrá un jaque mate en una rama del árbol si no hay movimientos posibles en ese nodo y hay algún jaque. Detectar un jaque es muchísimo más sencillo que detectar un mate. Asimismo también detectamos ahogamientos si no hay movimientos posibles y en el tablero no hay jaque. Este método puede simplificar la función de evaluación, la dificultad estriba en qué valores debe devolver el algoritmo en estos casos. La idea es que nosotros devolvemos un número más negativo (o positivo según se mire) si el jaque mate se produce cerca de la raíz del árbol, es decir, en el menor número de movimientos. Cuando hablamos de empates, esto es más complicado ya que dependiendo de la altura del juego a la que nos encontremos y de otros factores, tendrá una puntuación u otra. Observar que esto no es sencillo de programar y si no tenemos cuidado añadiremos tiempo de cómputo innecesario al algoritmo.

3.2.16. Conclusiones

En este apartado se han mostrado muchas técnicas que actualmente se usan en la implementación de juegos de ajedrez. Cada una de estas técnicas individualmente quizá no mejore mucho la poda alfa-beta. La verdadera potencia de los distintos métodos es mezclarlos. Hay distintas técnicas que juntas proporcionan gran potencia al algoritmo. Por ejemplo:

La matriz de transposición se puede aplicar a todos los algoritmos aunque es especialmente importante, por no decir fundamental, en la profundización iterativa o en la poda de transposición creciente.

Si hablamos de profundización iterativa, esta técnica mejora muchísimo si le añadimos la búsqueda de la variación principal.

La búsqueda de la quietud mejora mucho si usamos movimientos nulos en las extensiones.

Como sugerencia después de estudiar los distintos algoritmos, parece que una buena táctica es usar búsqueda de la variación principal con profundización iterativa; por supuesto necesitamos una tabla de transposición de un tamaño relativamente grande. Como tenemos la tabla podríamos usar la táctica de los movimientos *killer* para el orden de exploración de los nodos. En los nodos finales usaremos búsqueda de la quietud con movimientos nulos. Deberíamos realizar un estudio más en profundidad de los movimientos nulos para ver en qué situaciones podrían ser aplicados.

3.3. Tablero

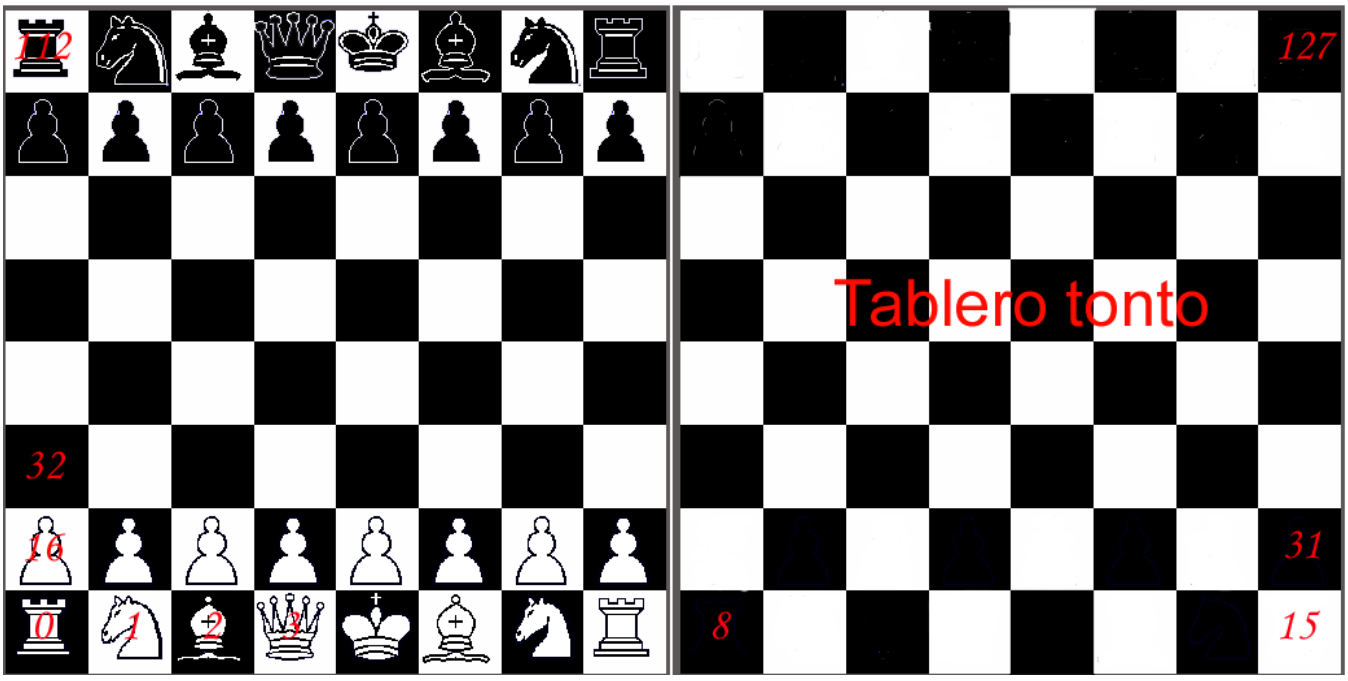
3.3.1.Representación del tablero

De todas las representaciones del tablero estudiadas, la usada para nuestro proyecto es la siguiente:

Se representa en un array de 128 casillas, que representa el tablero real a la izquierda y un tablero "tonto" a la derecha, es decir, a1 será la casilla 0, b1 la 1... h1 la 7 y a2 la 16. Para calcular una posición, se hace:

fila*16 + columna.

Podemos observar esta disposición en el siguiente gráfico:



Las ventajas de esta representación son las siguientes:

Comprobación de rangos: Si el rango se va por arriba (el número es mayor o igual que 128, el bit 8 va a estar a uno (tiene peso 128) y si el número es negativo, dada la representación en complemento a 2 también. Esto también pasaría para una representación de 64 bits de tablero (comprobando el bit 7), pero horizontalmente no es así y con la representación de 128 bits horizontalmente también se puede controlar el movimiento de esta forma, ya que si el bit 4 (de peso 8) está a uno, indica que estamos fuera de los límites del tablero.

Generación de movimientos: Con esto se generan los movimientos bastante rápido y de forma eficiente. Por ejemplo para generar los movimientos de un alfil, sería ir sumando o restando 17 ó 15 según la diagonal a observar. Cada pieza tendría un array de enteros que indicaría las posibles direcciones en que se puede mover para luego generar los posibles movimientos (para el alfil +17, +15, -15, -17). Habría que generar código especial para los enroques y peones, pero eso pasa siempre.

3.3.2.Tabla de transposición

Es una tabla hash donde se guardan las posiciones que se han ido analizando a lo largo de la partida, ya que es muy frecuente que haya que volver a analizarlas. Hay que definir unas claves para la hash distinta del índice que serían 768 números (6 piezas x 2 bandos x 64 casillas).Y a éstos hay que añadir 4 bits para indicar los permisos del enroque, 16 bits para indicar los permisos de comer al paso y uno que diga a quien le toca jugar.

Una clave de una posición se consigue haciendo la XOR de las claves de las 64 casillas. El índice se puede formar igual que la clave pero con una codificación distinta, o bien se puede obtener el índice a partir de unos cuantos bits de la clave, pero cuanto más grande sea la tabla hash más bits se tienen que sacar para el índice. Nosotros sacamos el índice de la clave.

Para ver si una posición esta en la hash se mira hash[índice], y si coinciden las claves, entonces se puede usar la información allí obtenida. Hay un mínimo riesgo de colisión que no se puede controlar. En caso contrario habrá que decidir si se reemplaza o no. Si se va a sacar el índice de la clave misma, es necesario usar por lo menos un entero de 64 bits para la clave, para dejar el riesgo de colisiones en despreciable.

Lo que se guarda y recupera en una hash para la búsqueda suele ser:

o.-Jugada recomendada
o.-Profundidad
o.-Valor del nodo
o.-Tipo de valor

La jugada recomendada es la que fue la mejor la última ocasión para esta posición o al menos causó un corte anteriormente. Es la primera jugada en probar esta ocasión.

El valor del nodo se refiere al valor resultante del nodo la vez pasada.

El tipo de valor señala si ese valor es exacto, o sólo un límite inferior o superior.

La profundidad señala cuanto le faltaba al nodo en medias jugadas para llegar a las hojas. Naturalmente sólo si ésta es mayor o igual que la profundidad actual se usa el valor guardado, si es que de algo sirve (en el mejor de los casos se retorna de inmediato, o por lo menos se agranda alpha o achica beta.)

Hay dos opciones para ver cuando se reemplaza algo:

- reemplazar siempre (se puede perder información).
- reemplazar cuando la profundidad nueva sea mayor que la vieja (puede haber información anticuada e inválida).

Una alternativa es usar dos tablas hash, una con cada criterio, y cuando se consultan para una situación, se consultan las dos tablas. Si una de las dos te permite podar se realiza la poda y en caso contrario se probaría el mejor movimiento de las dos antes de probar otros movimientos en el *minimax*.

Otra alternativa para el reemplazamiento de valores es tener un *flag* que diga si se puede sobrescribir o no la tabla hash; cuando acaba una jugada y la máquina mueve todos los valores del *flag* se ponen a uno, si se efectúa una lectura en la tabla el *flag* se pone a 0, si hay una colisión y el *flag* está a 1 se sobrescribe, si no se deja tal cual.

Nosotros en nuestro proyecto empleamos la alternativa de reemplazar siempre.

Veamos una posible forma de crear una tabla de transposición para nuestro programa.

Para crear una clave (la clave de la posición inicial) se empieza generando una clave inicial y luego se hace la XOR de los valores obtenidos en la tabla creada al inicio del programa con los tripletes (pieza, color, casilla) que correspondan. Luego posteriormente con cada movimiento simplemente hay que hacer la XOR de la clave anterior con el valor la pieza movida en su posición origen (para quitarla de la clave, ya que hacer la XOR dos veces es como no hacer nada) y la XOR de la clave resultante con el valor de la pieza movida en su posición destino.

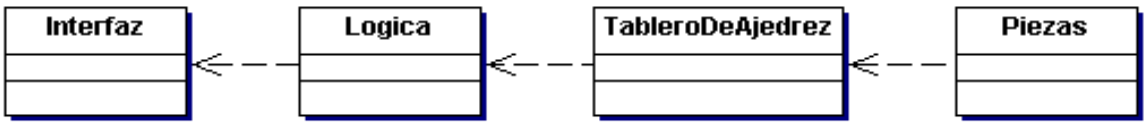
Para el índice se propone hacer la clave de la posición módulo el tamaño de la tabla. La tabla hash usada en nuestro proyecto tiene un tamaño de 262144 elementos.

4. Análisis

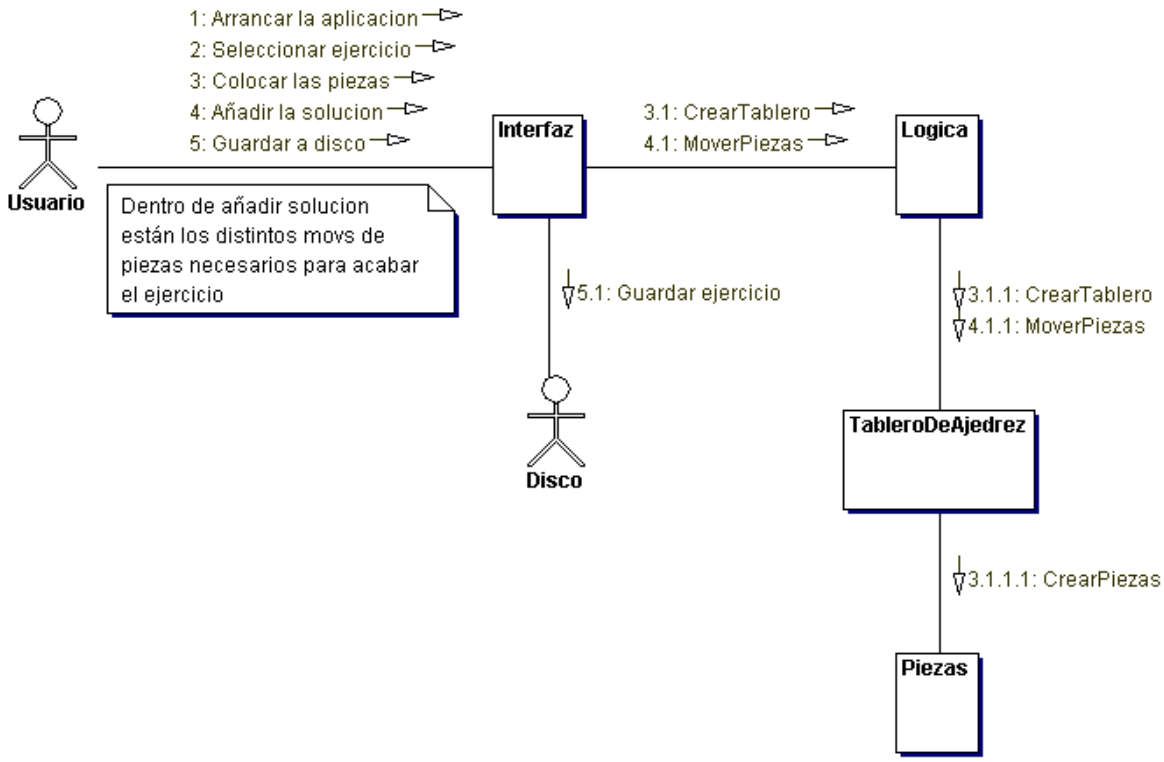
Pasamos ahora a explicar el análisis de nuestra aplicación. Primero se detallará el análisis de la aplicación del profesor.

4.1. Profesor

Vamos a mostrar el diagrama de clases de análisis:



Vemos ahora el diagrama de colaboración correspondiente:



Vamos a presentar las distintas clases y sus funcionalidades.

Interfaz: Interfaz de la aplicación, es la encargada de recoger todas las instrucciones que da el usuario y pasarlas a la lógica de la aplicación. Entre sus funcionalidades encontramos:

- Posibilidad de creación de ejercicios: El usuario va a poder crear distintos tipos de ejercicios del curso.

- Posibilidad de guardar estos ejercicios: El usuario debe disponer de la posibilidad de guardar los distintos ejercicios en el disco para su posterior uso.
- Gestión de la creación de ejercicio: La interfaz debe tener todas las opciones posibles para que el usuario pueda crear de manera sencilla todos los ejercicios disponibles del curso de ajedrez.

Lógica: Lógica del programa, es la encargada de comprobar que los ejercicios que crea el profesor son correctos. Es la encargada también de guardar los ejercicios en archivos y de gestionar la aplicación. Entre sus funcionalidades encontramos:

- Creación de ejercicios: Recoge los datos que le envía la interfaz, comprueba si son correctos y si lo son, crea el ejercicio y lo guarda en disco.
- Gestión de errores: si un ejercicio no es correcto hay que decir que no lo es y la razón.
- Guardar los ejercicios creados en disco: estos ejercicios se guardaran en el disco duro en un formato que permita a la aplicación del curso abrirlos.

TableroDeAjedrez: Clase que representara un tablero de ajedrez con piezas. Cada ejercicio que se cree tendrá un tablero con las piezas en sus posiciones iniciales. Sobre este tablero se realizaran movimientos o acciones para completar el ejercicio. Funcionalidades:

- Debe representar fielmente un tablero de ajedrez con todas sus características. Es decir, no debe dejar hacer movimientos incorrectos, etc.

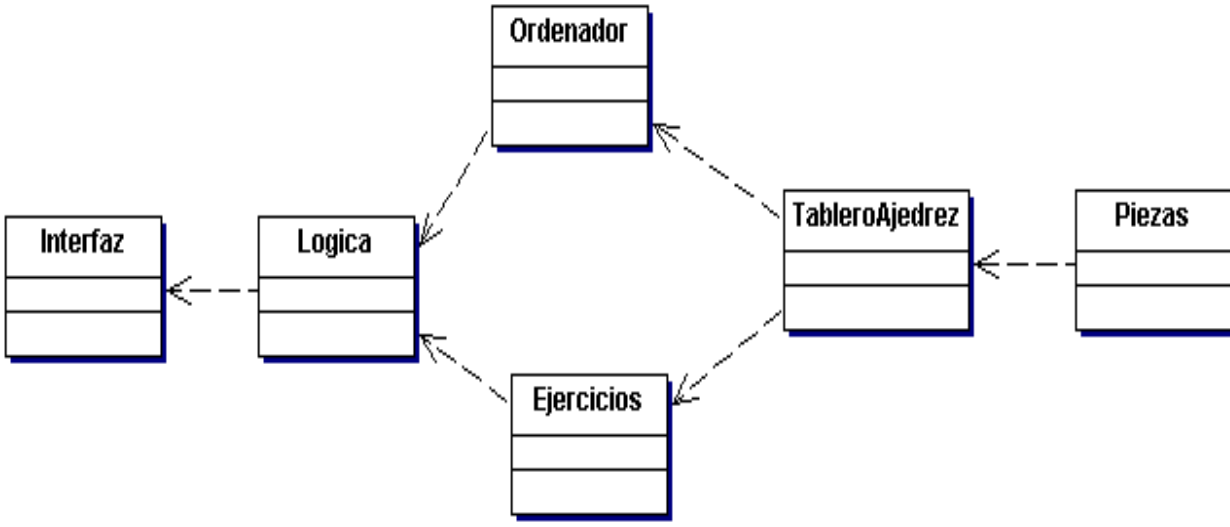
- Facilidades para la creación de los ejercicios. El tablero estará orientado a crear ejercicios, no estará orientado a jugar una partida.
- Debe dar soporte a todos los tipos de ejercicios.

Piezas: Clase que representa las distintas piezas que estarán dentro del tablero de ajedrez. Funcionalidades:

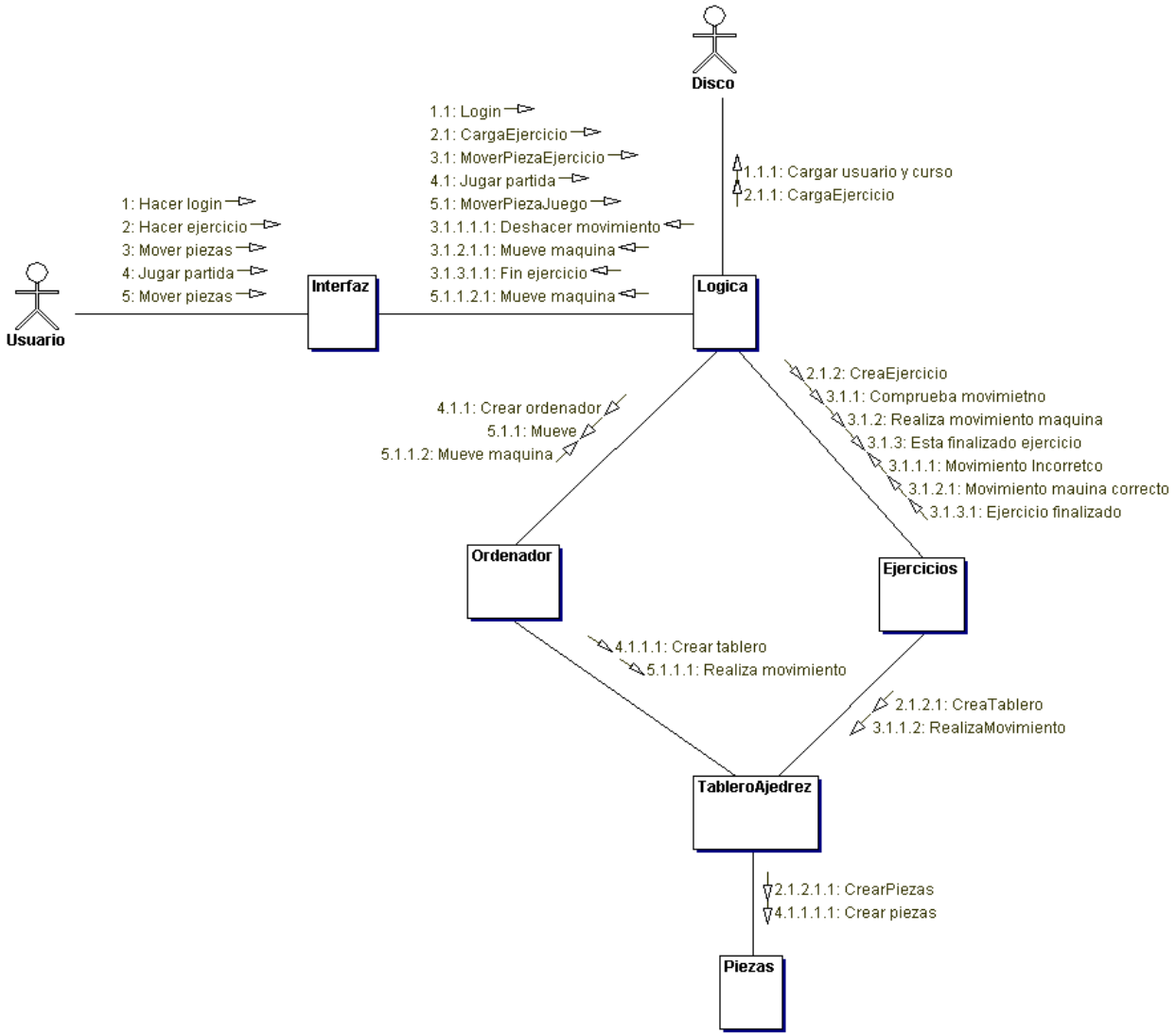
- En un primer momento cada pieza conoce su posición dentro del tablero.

4.2. Programa del curso

Diagrama de clases de análisis:



Vemos ahora el diagrama de colaboración correspondiente:



Vamos a presentar las distintas clases y sus funcionalidades:

Interfaz: interfaz de la aplicación, es la encargada de recoger todas las instrucciones que del usuario y pasarlas a la lógica de la aplicación. Debe ser una interfaz sencilla de usar por un usuario que nunca haya jugado al ajedrez. Funcionalidades:

- Posibilidad de realizar los ejercicios: debe dar al usuario la posibilidad de realizar los distintos ejercicios del curso, a su vez debe dar instrucciones al usuario sobre la realización de los ejercicios.
- Debe dar la posibilidad de jugar una partida completa de ajedrez.
- Debe dar soporte a todas las funcionalidades que la lógica de la aplicación permita de manera sencilla.

Lógica: Lógica de la aplicación que comunica la interfaz grafica con la gestión de los ejercicios y de la partida de ajedrez. Se encargará de filtrar todos las instrucciones que el usuario haga y de llevar a cabo las acciones necesarias para que éstas se cumplan, bien a nivel de lógica o pasándolo a las capas inferiores (ordenador, ejercicios). Funcionalidades:

- Gestión de ejercicios: Recibirá los movimientos que hace un usuario en un ejercicio determinado y los pasará a la clase ejercicios para comprobar que son o no correctos.
- Gestión de la partida: Contestará a cada movimiento del usuario pidiendo a la clase ordenador que devuelva un movimiento como contestación.

- Gestión del curso: Se encargará de cargar el curso completo y de ir pasando de un ejercicio a otro según el usuario vaya avanzando.
- Gestión de usuarios: Guardará información para cada usuario e irá actualizando esta información según el usuario vaya superando ejercicios.

Ordenador: "Inteligencia artificial" del programa, es la encargada de jugar una partida de ajedrez con el usuario. Se basará en algún algoritmo de búsqueda tipo *minimax*. Funcionalidades:

- Debe jugar de manera coherente y con un nivel relativamente bueno(ELO 900 aproximadamente)
- Tendrá varios niveles de dificultad.

Ejercicios: Clase que contendrá toda la funcionalidad de la realización de los ejercicios. Deberá contener todas las clases de ejercicios que estén en el curso. Funcionalidades:

- Gestionar la realización de los ejercicios: Debe decidir cuando un ejercicio está bien o mal y si mueve la máquina debe devolver el movimiento de la misma.

TableroAjedrez: Tendrá toda la información relativa al tablero de ajedrez, este tablero será el mismo tanto para los ejercicios como para la partida. Las funcionalidades de este tablero son distintas que las del tablero de la aplicación anterior por lo que puede ser distinto. Funcionalidades:

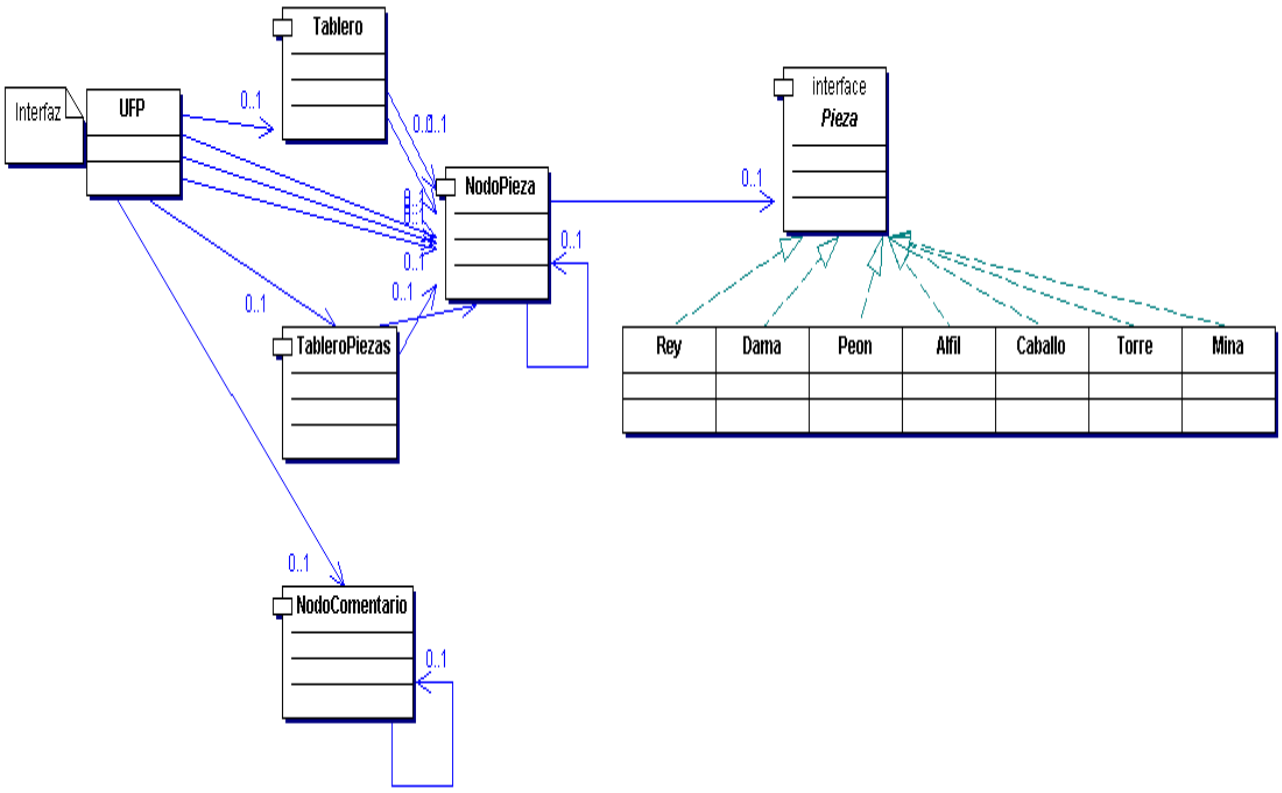
- Debe representar fielmente un tablero de ajedrez con todas sus características. Es decir, no debe dejar hacer movimientos incorrectos, etc.
- Dar las funcionalidades necesarias para que se pueda jugar una partida de ajedrez completa.
- Dar las funcionalidades necesarias para la realización de todos los ejercicios del curso.
- Piezas: Clase que representa las distintas piezas que estarán dentro del tablero de ajedrez. Funcionalidades:
- En un principio las piezas no conocen su posición dentro del tablero.
- Debe dar soporte a las distintas funcionalidades del tablero de ajedrez.

5. Diseño

Pasamos a explicar el diseño de la aplicación. Para ello vamos a comentar todas las clases de diseño obtenidas y sus principales funcionalidades y métodos. Empecemos por el diseño de la aplicación del profesor.

5.1. Profesor

Este es el diagrama de clases:



Vamos a presentar las distintas clases de la aplicación:

UFP: Es la clase de la interfaz gráfica de la aplicación. Se pretende que la interfaz sea una interfaz de ventanas, sencilla y fácil de usar.

En su interior se encuentra un tablero, éste es el tablero donde vamos a generar el ejercicio. También encontramos un objeto de la clase NodoComentario donde vamos a guardar los distintos comentarios de la aplicación. Otro objeto que va a tener va a ser uno de la clase TableroPiezas donde van a estar las distintas piezas que vamos a poder añadir a nuestro proyecto. Además vamos a tener varios objetos de la clase NodoPieza donde vamos a guardar los distintos movimientos de los ejercicios y nos servirán para deshacer el último movimiento. Por último tendremos muchas variables para controlar las distintas funcionalidades del programa y para controlar distintos aspectos de la interfaz gráfica.

Principales métodos:

- GLScene: Dibuja la interfaz por pantalla. La interfaz está programada en *OpenGL*.
- OnMouseDown: Captura la pulsación del ratón sobre el formulario. Es importante porque con este método vamos a saber cuál es la pieza sobre la que se está pinchando.
- OnMouseUp: Se ejecuta cuando se deja de pulsar el botón izquierdo del ratón. Este método probablemente sea el más importante de la aplicación. Aquí controlamos en qué casilla se coloca la pieza y, también, se crean los distintos movimientos que luego se van a guardar en el ejercicio.
- GuardarArchivo: Este método guarda en un archivo de texto el ejercicio en un formato determinado que luego pueda ser reconocido por la aplicación del curso de ajedrez para poder abrirlo.

Tablero: Esta clase representa un tablero de ajedrez. La información se guarda como dos listas de piezas (blancas y negras), cada pieza conoce su posición. También vamos a encontrarnos con una variable que nos indica el turno de juego y variables que controlan la parte gráfica (texturas, etc.)

- Tablero: Constructor, se le introducen como parámetro dos objetos de tipo NodoPieza que son la lista de blancas y negras así como dos enteros, uno para el turno y el otro para la textura a aplicar en el tablero.
- Dibuja: Todas las clases que tengan algún componente que se deba dibujar por pantalla van a tener este método. Este método dibuja un tablero por pantalla con las piezas que tiene ese tablero.
- DevuelvePieza: Dada la posición y el color (no es obligatorio) de la pieza me devuelve la pieza que hay en esa posición.
- PermitidoMov: Me dice si el movimiento de una pieza es legal o no.
- Pon: Coloca una pieza nueva en el tablero en una posición determinada.
- BorraPieza: Borra del tablero la pieza que se encuentre en la posición indicada
- Distintos métodos que pueden ser usados en algún ejercicio para comprobar ciertas características del tablero.

TableroPiezas: Representa las piezas que van a aparecer por pantalla a los lados del tablero de ajedrez. La idea es que el usuario elija las piezas de aquí y las coloque encima del tablero "pinchando y arrastrando". En esta clase nos vamos a encontrar con dos objetos de la clase NodoPieza que son dos listas de piezas (blancas y negras) así como variables que controlan el número de piezas de cada tipo que se han introducido en el ejercicio.

- Dibuja
- DevuelvePieza: Me dice la pieza que se encuentra en la posición en la que he pinchado.
- Distintos métodos que controlan el número de piezas de cada tipo que se han introducido en el tablero.

NodoPieza: Esta clase es una lista de piezas, se usa para representar las piezas que se encuentran en un tablero. La lista será una lista dinámica, por lo que la clase lo que realmente representa es un eslabón. Por dentro la clase lo que tiene es un puntero a una pieza y un puntero al siguiente NodoPieza.

- NodoPieza: Vamos a tener tres constructores, el primero sin ningún parámetro que me crea una lista de piezas vacía. El segundo de un parámetro, una pieza, que crea una lista con una única pieza. Y por último un constructor de dos parámetros, una pieza y un NodoPieza, que crea una lista de piezas con la primera pieza siendo la que se mete como parámetro y a continuación el resto de la lista que también metemos como parámetro.
- GetPieza: Devuelve la pieza

- GetSig: Devuelve el siguiente elemento de la lista.
- SetPieza: Introduce una pieza en la cadena.
- SetSig: Introduce el siguiente elemento de la cadena, tenemos dos posibilidades, introducir una pieza o introducir un NodoPieza

NodoComentario: Representa una lista de comentarios del ejercicio. Esta lista va ser dinámica. Vamos a tener un *AnsiString* que representa al comentario y un puntero al siguiente elemento de la lista.

- Los métodos son los mismos que en NodoPieza.

Pieza: Clase virtual que representa cualquier pieza del ajedrez. Va guardar la posición de la pieza en el tablero mediante dos enteros, fila y columna. Va a guardar también la posición dentro del formulario donde se dibuja así como el color de la pieza. También va a tener distintas variables que nos permitan dibujar la pieza.

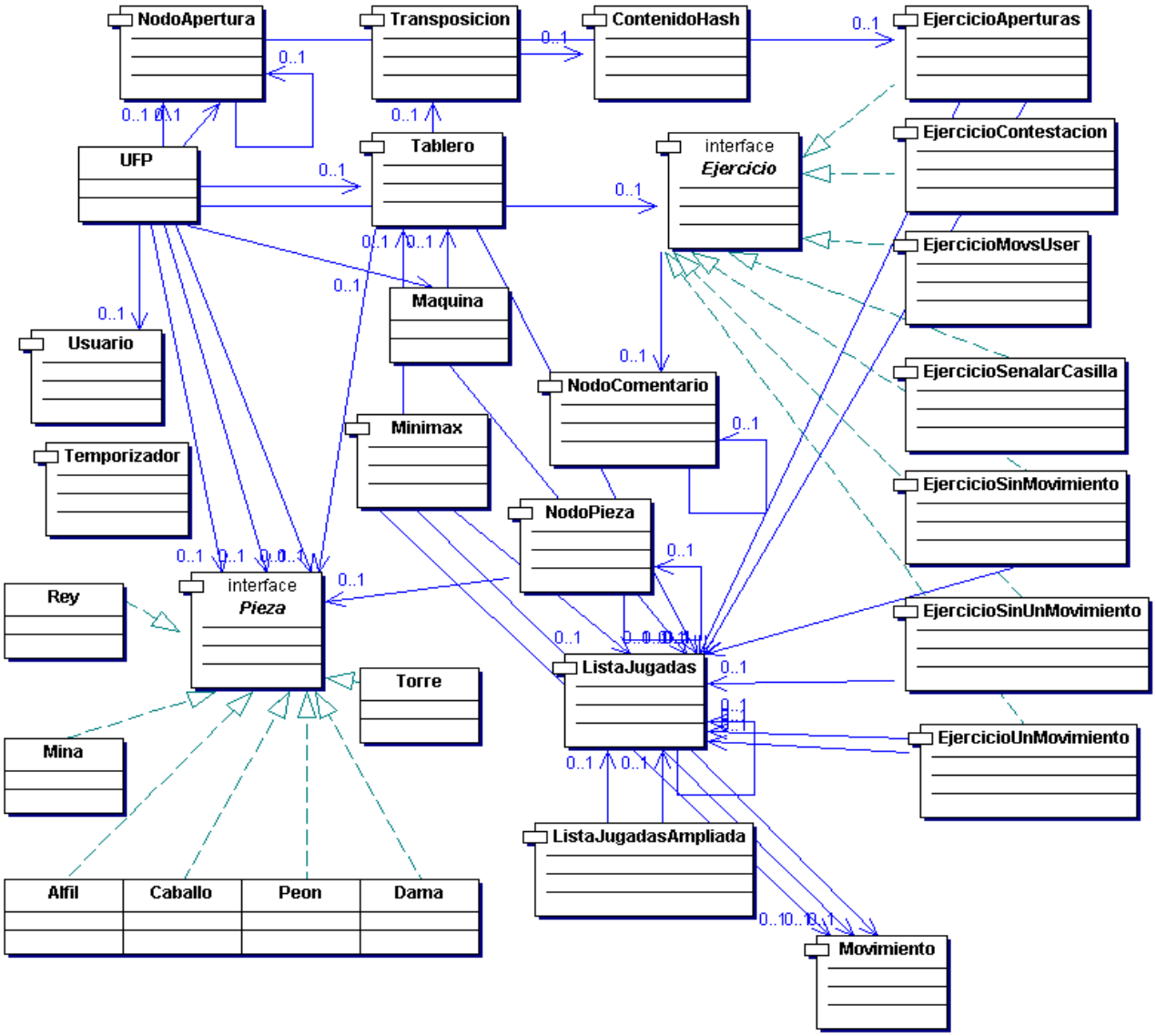
- Pieza: Constructor, le introducimos la posición dentro del tablero y el color de la pieza.
- PermitidoMovimiento: Método virtual que me dice si una pieza se puede mover de una posición a otra del tablero.
- QuienSoy: Clase virtual que me devuelve un entero indicándome el tipo de pieza que soy. 0-Rey, 1-Dama, etc.
- Dibuja
- Métodos que acceden y modifican todas las variables de la pieza.

Rey, Dama, Peón, Alfil, Caballo, Torre: Clases que heredan de Pieza y representan a las distintas piezas del ajedrez.

Mina: También hereda de pieza y representa un obstáculo dentro del tablero de ajedrez que se va a usar en determinados ejercicios.

5.2. Programa del curso

Diagrama de clases:



Vamos a presentar las distintas clases de la aplicación:

UFP: Es la interfaz gráfica de la aplicación. Debe permitir realizar los distintos ejercicios y también debe permitir disputar una partida de ajedrez. Vamos a tener dentro un tablero, que va a representar el tablero donde se van a realizar los distintos tipos de ejercicio. También vamos a tener un objeto de la clase Maquina, esta clase representa al contrincante en una partida de ajedrez. Vamos a encontrar un objeto Ejercicio que es el ejercicio que actualmente se está realizando. Tenemos un objeto de tipo Minimax que es el árbol de juego necesario para que la máquina piense y juegue bien al ajedrez. Podemos ver también un objeto de la clase NodoAperturas, que guarda las distintas aperturas que tenemos para que juegue la máquina o para realizar un ejercicio de aperturas. Vamos a tener dos listas dinámicas de Pieza para dibujar las piezas comidas en una partida de ajedrez. Por último tenemos un objeto de la clase Usuario que es el usuario que ha hecho *login* con sus características. Dentro de esta clase también tenemos distintas variables que controlan las distintas funcionalidades del programa y que controlan que se dibuje todo correctamente. Métodos:

- GLScene: Dibuja el formulario y todos los objetos que se deben mostrar por pantalla.
- CargarEjercicio, CargarCurso, etc.: Distintos métodos que cargan desde un fichero los distintos datos que necesita la aplicación: el usuario, un ejercicio, una apertura, etc.
- JugarEjercicioTipo4, JugarEjercicioTipoNo4, etc.: Permiten jugar los ejercicios del curso.
- OnMouseDown, OnMouseUp: Controlan el click del ratón. En estos métodos es donde se comprueba si un movimiento es correcto, llaman a los distintos métodos de jugar ejercicios. También en el segundo método se llama a la función que hace que la máquina piense un movimiento en una partida.

- Distintos métodos que sacan comentarios, formularios, etc. por pantalla.

Usuario: Clase que representa al usuario que hace *login* y que está realizando el curso. Tendrá el nivel del curso en el que se encuentra así como estadísticas de número de ejercicios correctos etc.

- Usuario: Constructor, como parámetros se introducen en qué nivel del curso se encuentra (capítulo y número de ejercicio) así como las estadísticas de ejercicios correctos, realizados, etc.
- Métodos que acceden y modifican todas las variables del usuario (capítulo, número de ejercicios realizados, etc.)

Temporizador: Clase que se usó en la fase de elaboración del algoritmo de juego de la máquina para medir el tiempo que tardaba el ordenador en dar una respuesta.

NodoApertura: Lista dinámica de la clase EjercicioAperturas que contiene todas las aperturas que se pueden realizar por la máquina o por el usuario. Contiene un puntero al ejercicio de aperturas y otro puntero al siguiente ejercicio.

- NodoApertura: Cuatro constructores, uno sin parámetros que construye una lista vacía. El segundo con un parámetro, EjercicioAperturas, que construye una lista con un elemento. El tercero tiene un parámetro, NodoApertura, y es el constructor copia. Y por último un constructor con dos parámetros, el primero es un EjercicioAperturas y el segundo un NodoApertura. Construye la lista cuyo primer el elemento es el parámetro que hemos introducido y pone detrás la lista que también metemos como parámetro.

- Métodos get y set de los distintos elementos de la lista.

Tablero: Es el tablero donde se van a realizar los distintos ejercicios. Contiene una lista de 128 piezas que es el tablero de juego. También tenemos información redundante de este tablero que es una matriz de 8x8 de enteros, esto lo usamos para mejorar la eficiencia. Tenemos distintos arrays donde se guardan los movimientos de determinadas piezas. Podemos encontrar distintas variables para controlar el enroque y el comer al paso así como variables que controlan donde se encuentran los reyes y el número de piezas de cada color. También posee un objeto del tipo Transposición que es la matriz de transposición que se usa para que la máquina juegue.

- Tablero: Hay tres constructoras, la primera crea un tablero con las piezas en la posición inicial. El segundo crea un tablero pero las piezas que hay se introducen como parámetro. Y por último el tercer constructor es un constructor copia.
- GetCasillas: Me devuelve la pieza que se encuentra en la posición del tablero que introduzco como parámetro. Esta posición es un índice de la lista donde guardo las piezas.
- DevuelvePieza: Igual que el método anterior pero esta vez introduzco como parámetro fila y columna.
- Mueve: Método que mueve la pieza que se encuentra en el índice que introducimos como parámetro al destino que también introducimos como parámetro. Actualiza tanto la lista de piezas como la matriz de enteros con las posiciones de las piezas.

- BorraPieza: Borra la pieza que se encuentra en la posición que le indicamos.
- Dibuja
- HacerMovimiento: Se le introduce como parámetro un objeto de la clase movimiento y este método realiza el movimiento si es correcto. Usa al método Mueve.
- DeshacerMovimiento: Deshace el último movimiento realizado sobre un tablero.
- ObtenerMovimiento: Me devuelve una lista de movimientos con todos los movimientos posibles de un jugador en el tablero.
- EsMate, EsJaque: Métodos que me dicen respectivamente si en el tablero hay jaque o jaque mate.
- Distintos métodos que acceden y modifican las distintas variables del tablero, así como métodos que realizan la coronación y el comer al paso. También encontramos métodos que controlan las texturas que se usan para dibujar el tablero.

Maquina: Representa al contrincante del usuario en una partida de ajedrez. Tenemos dentro un objeto de la clase Minimax que es el árbol de juego del programa y un entero que representa la profundidad de este árbol.

- Maquina: Construye una máquina.
- Juega: Devuelve el mejor movimiento del ordenador para un tablero determinado que metemos como parámetro.

Minimax: Clase que representa un árbol Minimax con poda alpha-beta. Tenemos dentro un tablero que es el tablero donde se va a realizar la búsqueda alpha-beta, una lista de jugadas que es la lista de jugadas inicial sobre la que realizar la búsqueda. También tenemos variables que controlan el turno y la profundidad del árbol.

- Minimax: Construye un árbol *minimax* vacío donde sólo se inicializan las variables.
- Juega: Esta función es la que inicializa la construcción del árbol de búsqueda.
- AlphaBeta: Función recursiva que va a ser la función principal de la clase. Se llama recursivamente según vaya profundizando niveles dentro del árbol. Usa poda alpha-beta con una matriz de transposición, jugadas *killer* y búsqueda de la quietud. También puede jugar con profundización iterativa y búsqueda de la variación principal.
- Quietud: Función que extiende el árbol en los nodos finales más conflictivos para buscar si determinadas jugadas son o no buenas.
- FuncionEvaluacion. Función que devuelve un float que representa la valoración de un tablero determinado. Esta función tiene en cuenta muchos aspectos del tablero: posicional, número de piezas, etc.
- Funciones auxiliares de la función de evaluación que evalúan cada tipo de piezas por separado.

NodoPieza: Lista dinámica de piezas; contiene un puntero a la pieza y otro puntero al siguiente elemento de la lista.

- NodoPieza: Cuatro constructores, uno sin parámetros que construye una lista vacía. El segundo con un parámetro de tipo Pieza, que construye una lista con un elemento. El tercero tiene un parámetro de tipo NodoPieza, y es el constructor copia. Y por último un constructor con dos parámetros, el primero es una Pieza y el segundo un NodoPieza. Construye la lista cuyo primer elemento es el parámetro que hemos introducido y pone detrás la lista que también metemos como parámetro.
- Métodos get y set de los elementos de la lista.

Movimiento: Clase que es un movimiento dentro del tablero. Tiene el índice inicial y el índice destino del movimiento. También tenemos dos parámetros, tipoPieza y color, que usaremos para decir qué pieza nos hemos comido en un movimiento (si nos hemos comido alguna) para que luego podamos deshacer el movimiento.

- Movimiento: Constructor con todos los parámetros de la clase y dos booleanos que me indican si el movimiento es un enroque o comer al paso.
- Métodos set y get de todos los parámetros de la clase.

ListaJugadas: Lista dinámica de movimientos que usaremos a la hora de generar el árbol de juego para guardar todos los movimientos posibles de un tablero. Tenemos un puntero al movimiento y otro al siguiente elemento de la lista. Es necesario comentar aquí también la clase listaJugadasAmpliada, que es una extensión de ésta y que usamos para la generación de movimientos. Esta última clase tiene un puntero al primero y otro al último elemento de ListaJugadas.

- ListaJugadas: Cuatro constructores, uno sin parámetros que construye una lista vacía. El segundo con un parámetro de tipo Movimiento, que construye una lista con un elemento. El tercero tiene un parámetro de tipo ListaJugadas, y es el constructor copia. Y por último un constructor con dos parámetros, el primero es de tipo Movimiento y el segundo de tipo ListaJugadas. Construye la lista cuyo primer elemento es el parámetro que hemos introducido y pone detrás la lista que también metemos como parámetro.
- AnadirMovimiento: Añade un movimiento al principio de la lista.
- Métodos get y set de todos los elementos de la lista.

Pieza: Clase virtual que representa las distintas piezas que encontramos en una partida de ajedrez. Guardamos dentro el color de la ficha. En esta ocasión la ficha no sabe en qué posición del tablero se encuentra. También guardamos variables que controlan las texturas que se usan con las fichas.

- Pieza: Constructor que le meto como parámetro el color de la pieza.
- QuienSoy: Método virtual que me devuelve un entero diciéndome el tipo de pieza que soy.
- Dibuja

Rey, Dama, Peón, Alfil, Caballo, Torre: Clases que heredan de Pieza y representan las piezas del ajedrez.

Mina: Clase que hereda de Pieza y representa un obstáculo en el tablero de determinados ejercicios.

Transposición: Clase que representa la matriz de transposición que se usa para que la máquina juegue. Tiene un array de 262144 posiciones de ContenidoHash que es la verdadera matriz de transposición. En esta matriz guardamos tableros y su valoración. Exactamente no se guardan tableros, sino códigos que representan al tablero.

- EstaEnTabla: Nos dice si una posición está guardada en la matriz. Se le pasa el código del tablero y una profundidad (ya que si está guardada la posición, pero se ha explorado a partir de ella menos de lo que se quiere explorar, no nos vale)
- GetCodigo: Se crea el código de una posición cualquiera pasada en casillas. (Se usa para cuando se carga una partida)
- DevolverValor: Se devuelve el valor de una posición si está en el tablero y es válido su valor.
- GuardarTabla: Se almacena una posición en la tabla, con su código, profundidad, valor y tipo (si se trata de un valor real o es una estimación alfa o beta)

ContenidoHash: Contenido de una posición de la matriz de transposición. Tenemos el código hash del tablero, la profundidad donde se encontró el tablero, el tipo del mismo y la valoración.

Ejercicio: Clase virtual que representa un ejercicio del curso de ajedrez. Tiene dentro un entero que representa el color de las piezas del usuario, el turno de juego, el tipo de ejercicio y una lista de comentarios.

- Ejercicio: Constructor donde se introducen todos los parámetros de la clase.
- GetComentario: Devuelve el primer comentario de la lista de comentarios.

- SiguienteComentario: Pasa al siguiente comentario de la lista.
- EstaAcabado: Función virtual que me dice si está acabado el ejercicio.
- SiguienteMovimiento, SiguienteMovimientoUser, SiguienteMovimientoMaquina: Función virtual que me devuelve el siguiente movimiento del ejercicio si lo hubiera.
- ComprobarMovimiento, ComprobarMovimientoUser, ComprobarMovimientoMaquina: Función que comprueba si un movimiento de la máquina o el usuario es correcto.
- EjercicioAperturas, EjercicioContestacion, etc.: Todas estas clases heredan de ejercicio y representan todos los tipos de ejercicios que hay en el curso. Cada una de ellas tiene atributos diferentes dependiendo de las necesidades del ejercicio.

6. Desarrollo del curso y planificación del mismo

6.1. Introducción

El objetivo de este apartado es poner por escrito los contenidos del proyecto que estamos realizando. Estos contenidos son todas las funcionalidades que creemos que debería tener un programa de ajedrez como el nuestro. En este documento vamos a incluir todas las mejoras que se nos han podido ocurrir en determinados momentos y que no deben estar necesariamente acabadas sino que puede estar planificado hacerlas en un futuro.

6.2. Curso de ajedrez

Lo primero que se hizo fue generar un curso de ajedrez con contenidos didácticos para que una persona sin conocimientos previos de ajedrez pudiese llegar a aprender a jugar al mismo. Este curso se dividía en distintos capítulos, cada capítulo trataba de un tema que nosotros creímos fundamental en el aprendizaje de ajedrez. Este curso se debía incluir en un programa de manera secuencial para que un alumno pudiese aprender a jugar al ajedrez. Ya teníamos entonces el programa principal de nuestro proyecto; un programa que incluyese un curso de ajedrez que permitiese a un alumno realizar este curso de manera secuencial, realizando ejercicios que nosotros incluiríamos. Teniendo ya claro esto, surgió otra idea: ¿por qué no permitir que alguien edite estos ejercicios?. Es decir, una especie de profesor que crea los ejercicios del curso de ajedrez y los guarda para que el alumno los vaya realizando. Aquí surgió la segunda aplicación del proyecto; un programa "profesor" que permitiese editar y crear los distintos tipos de ejercicios para que luego la otra aplicación leyese estos ejercicios y permitiese al alumno realizarlos. Pasamos a explicar ahora el contenido del curso de ajedrez.

Primero dividimos el curso en capítulos secuenciales empezando desde los conceptos básicos hasta los más avanzados; dentro de cada capítulo definimos una serie de ejercicios ejemplo que mostraban cómo deberían ser los ejercicios de ese capítulo. Esto como documentación está muy bien, pero lo que realmente necesitábamos era buscar una serie de rasgos comunes en los ejercicios para conseguir separarlos en clases distintas para poder ser codificados. Una vez separados nos dimos cuenta que había ejercicios en los que la máquina necesitaba mover, pero no realizar un movimiento predefinido, sino una partida real de ajedrez o, al menos, unos cuantos movimientos coherentes. Aquí surgió la tercera aplicación del proyecto; un programa que jugase al ajedrez.

6.3. Programa profesor

6.3.1.Funcionalidades básicas

Básicamente esta aplicación debería incluir un editor de ejercicios de ajedrez que permitiese a un profesor situar las piezas en una posición inicial, definir una serie de movimientos (si lo desea) y clasificar el ejercicio. Se decidió que todo esto se haría de manera gráfica con una interfaz sencilla, intuitiva y agradable a la vista. Esta interfaz deberá facilitar la labor del profesor en la creación de los ejercicios.

INTERFAZ:

Deberá tener un tablero de ajedrez vacío donde situaremos las distintas piezas. Estas piezas se situarán a ambos lados del tablero en sendas columnas y el profesor pinchará y arrastrará las piezas hasta situarlas en el tablero. Se incluirá una ayuda para el profesor: como crear paso a paso un ejercicio, objetivo del ejercicio, etc. Esta ayuda podrá ser dinámica (en cada instante se dirá al profesor lo que debe ir haciendo) o estática (se pedirá la ayuda en un momento determinado). La interfaz gráfica deberá poner a disposición del profesor, bien en forma de menús o botones, toda la funcionalidad de la aplicación de manera sencilla e intuitiva.

TIPOS DE EJERCICIOS:

Los ejercicios se dividirán dependiendo de si la máquina debe mover, si es el usuario o si son ambos los que tienen que mover, y de si hay que realizar un movimiento o varios. Habrá ejercicios especiales (aperturas, juego medio) que formaran clases distintas. Para cada ejercicio se podrá generar un tablero inicial sobre el que realizar el ejercicio. Para determinados ejercicios el profesor podrá definir los movimientos que hay que realizar tanto por el usuario como por la máquina, esto se hará de manera gráfica pinchando y arrastrando la pieza que se moverá en cada caso. Se deberá poder deshacer el último movimiento realizado, por si el profesor se ha equivocado al realizarlo.

SALIDA DE LA APLICACIÓN:

Todos los ejercicios realizados por el profesor deberán guardarse de algún modo en disco para que puedan ser abiertos por la otra aplicación. En primer lugar esto se realizará usando archivos.

6.3.2.Funcionalidades extras

Inclusión de una base de datos para guardar los ejercicios que cree el profesor. Estos ejercicios creados admitirán distintos formatos de salida a elegir por el profesor. Se dará soporte a guardar el ejercicio en látex para ser consultado desde otro visor. También se generará código para crear un *applet* con el ejercicio y que pueda sea ejecutado en red.

Los ejercicios incluirán comentarios del profesor, estos comentarios se mostrarán al alumno mientras vaya haciendo movimientos y deberán ser incluidos por el profesor mientras crea el ejercicio.

El profesor podrá editar y crear un nuevo curso de ajedrez, no sólo crear ejercicios para un curso ya creado sino crear el curso él mismo.

6.4. Programa del alumno

6.4.1. Funcionalidades básicas

Este programa deberá incluir los ejercicios que se hayan generado desde el profesor para que puedan ser ejecutados aquí y probados por un alumno.

INTERFAZ GRAFICA

Deberá incluir un tablero de ajedrez con piezas que el alumno moverá pinchando y arrastrando sobre el tablero. La interfaz deberá ser agradable a la vista, sencilla y de fácil manejo. Esta interfaz deberá incluir soporte para mostrar los comentarios del ejercicio, además de mostrar los distintos movimientos del usuario y de la máquina con la nomenclatura del ajedrez. También se deberá poder mostrar el progreso del alumno a lo largo del curso, con los capítulos superados, el que se está realizando y los que quedan por hacer.

ALUMNO

Se deberá poder crear nuevos alumnos y la aplicación podrá tener distintos alumnos que pueden ir por distintas partes del curso. Los alumnos deberán hacer *login* en la aplicación y se permitirá que un alumno cambie su contraseña. Se incluirán estadísticas de ejercicios resueltos y número de fallos para poder valorar el nivel de juego de un alumno de manera numérica. Estas estadísticas se irán modificando a lo largo del curso y al salir del programa deben guardarse en disco.

CURSO

El curso de ajedrez deberá estar creado, los ejercicios se generan desde el profesor pero la estructura del curso es fija y creada de antemano. Un alumno no podrá realizar ejercicios que no se correspondan con su nivel, es decir, no podrá hacer ejercicios de un capítulo si no ha superado todos los anteriores, lo que sí podrá hacer es repetir ejercicios de capítulos anteriores.

PARTIDAS

El alumno podrá jugar partidas de ajedrez en el momento del curso que desee, tendrá 2 opciones. Jugar una partida con un nivel que se ajuste a su nivel de juego o bien jugar una partida en la que el alumno elija el nivel de juego. Los niveles del juego estarán prefijados de antemano.

AYUDA

El programa deberá incluir una ayuda completa de manejo para el alumno, explicando paso a paso cómo funciona la aplicación. También se incluirá la posibilidad de solicitar a la máquina en mitad de una partida consejo sobre el siguiente movimiento a realizar por nosotros, nos deberá decir cual es el mejor movimiento y por qué. También podremos solicitar a la máquina que nos dé su valoración de una posición determinada de una partida, diciéndonos las ventajas e inconvenientes de la situación de las piezas que tenemos en ese momento.

6.4.2.Funcionalidades extras

Incluir conexión con una base de datos donde se encontrarán los ejercicios y los distintos usuarios. Permitir al usuario que guarde una partida a mitad del juego para luego poder continuar jugando en ese momento. Incluir guardar partidas en formatos distintos, *látex, pdf*, etc, para poder ser exportadas. Poder generar un *applet* con una partida para que se pueda jugar en red.

6.5. Programa de ajedrez

6.5.1. Funcionalidades básicas

Crear un programa que juegue al ajedrez con un nivel cercano al de un usuario que está aprendiendo a jugar. Se aplicarán técnicas que usan programas comerciales para conseguir un juego de ajedrez que juegue medianamente bien. Estas técnicas incluyen (poda alfa-beta, matriz de transposición, función de evaluación) Con ello se pretenderá llegar a un nivel ELO cercano a 800, 900 que estimamos que es el nivel que podrá conseguir un alumno al terminar el curso de ajedrez. La aplicación deberá ser lo suficientemente rápida para que en el peor de los casos de una respuesta en menos de 1 minuto aproximadamente. Deberá incluir todas las características especiales del ajedrez: enroque, comer al paso y coronación. De este modo, no deberá generar movimientos incorrectos y tampoco permitir al jugador realizar los mismos. En sí se programará como un programa independiente pero en realidad será parte de la aplicación del alumno, por lo que compartirá su interfaz gráfica y todas sus funcionalidades.

La función de evaluación merece un estudio aparte. En ella, además de incluir criterios para valorar una posición del tablero, se deben incluir comentarios que se van a mostrar al alumno explicando las razones de la valoración del tablero. Estos comentarios serán del tipo hay un caballo cerca del centro y amenazando a una pieza, la dama está dando jaque, etc. Todos estos comentarios deben ser claros y sencillos ya que los leerá un alumno que no está acostumbrado a jugar al ajedrez.

La máquina podrá jugar a distintos niveles, desde el más sencillo, mover al azar, hasta el más completo, pasando por una serie de niveles intermedios que estarán definidos de antemano. Por ejemplo nivel al azar, nivel que siempre intenta comer, nivel que evalúa la dama y el rey, etc.

6.5.2.Funcionalidades extras

Incluir técnicas más sofisticadas para que la máquina juegue mejor al ajedrez (búsqueda de la quietud, variación principal, etc) con ello se pretende alcanzar un nivel ELO cercano a 1200.

Modificar los niveles del juego para incluir una valoración numérica de cómo juega la máquina, es decir, si la evaluación va de 0 a 1, 0 sería mover al azar y 1 sería jugar en el nivel más difícil. El 0.3 sería por ejemplo un nivel con la función de evaluación reducida a evaluar unas pocas piezas y una profundidad del árbol de exploración de 1 y así para el resto de niveles, es decir, que los niveles de dificultad no estén prefijados sino que dependan de un valor numérico.

7. Implementación

7.1. Introducción

En este documento vamos a recoger lo que realmente se programó de todo lo que se dijo en la planificación y las cosas que han quedado pendientes.

7.2. Programa del profesor

La interfaz gráfica se realizó. Esta interfaz gráfica consta de un tablero de ajedrez y piezas a los lados del tablero. El profesor arrastra estas piezas sobre el tablero para colocarlas. Se clasificaron todos los tipos de ejercicios comentados en la especificación. Los ejercicios de aperturas no se pueden generar, se consideró que era mejor tener los ejercicios de aperturas ya creados. Se incluyen las 19 aperturas que se comentaron en la especificación. El profesor elige el tipo de ejercicio que quiere crear y la máquina mediante una ayuda por pantallas le va guiando en la creación del ejercicio. Se incluyó el soporte para los comentarios del profesor en cada movimiento; el profesor antes de realizar el movimiento escribe el comentario que quiere que le salga al alumno cuando realice el movimiento. Se ha incluido un complejo tratamiento de errores en los ejercicios, por ejemplo no se pueden poner dos alfiles del mismo color en la misma diagonal, no se pueden poner más piezas de las permitidas por el ajedrez, etc.

No se incluyó la base de datos, todos los ejercicios se guardan en archivos de texto. El formato del archivo se ideó de manera que desde la aplicación del alumno fuese sencillo leer un ejercicio. El nombre de los archivos y la parte del curso a la que pertenecen se gestionan de manera automática por el programa.

Se desechó la posibilidad de exportar el ejercicio en distintos formatos (*látex, applets*) debido a las complicaciones que tenía hacerlo y a que entonces no hubiésemos podido hacer algunas funcionalidades que creíamos más relevantes.

Vamos a comentar aspectos del código que consideramos importantes.

Se consideró que la mejor manera de representar un tablero de ajedrez era mediante dos listas de piezas, blancas y negras. Las piezas conocían su posición dentro del tablero. Esto se decidió porque simplificaba mucho el guardar un ejercicio en disco. Los comentarios de los ejercicios se guardan en listas dinámicas de comentarios.

7.3. Programa del alumno

Separamos la parte que juega al ajedrez del curso propiamente dicho. Aquí vamos a explicar el curso.

La interfaz gráfica consta de un tablero de ajedrez con cuadros de texto a los lados donde se van a mostrar los comentarios del profesor. Este programa admite todos los ejercicios que genera el profesor.

También se implementó una gestión de los usuarios que están realizando el curso. El usuario antes de empezar la aplicación debe hacer *login* en la misma. Se pueden crear nuevos usuarios. El control de los ejercicios resueltos por cada usuario y el punto del curso en el que se encuentra cada usuario se gestionan automáticamente.

Se programaron todos los tipos de ejercicios que se comentan en el documento del curso de ajedrez. Mencionar ciertas modificaciones en el orden de algún ejercicio (finales y juego medio). Y también destacar que las aperturas aunque están dentro del curso y son un tipo de ejercicio no las genera el profesor. Habrá un número limitado de aperturas predefinidas. Todas estas aperturas incluyen comentarios de cada movimiento y las razones del mismo. Se incluye la posibilidad de que el usuario solicite ayuda en un momento determinado de la apertura para que la máquina le indique el siguiente movimiento de la apertura.

Se implementaron todas las características especiales del juego de ajedrez. No sólo que las piezas se muevan correctamente, sino una serie de movimientos avanzados como son el comer al paso, el enroque y la coronación.

Los ejercicios del curso se cargan desde ficheros de texto ya que el profesor los guarda en este formato.

Mencionar que dentro de una partida el usuario puede tanto deshacer el último movimiento que ha hecho como guardar la partida en cualquier punto para luego volverla abrir si lo desea.

No se incluyó la exportación en otros formatos (*látex*, etc) del tablero de ajedrez.

7.4. Programa de ajedrez

Este programa es el que juega las partidas de ajedrez. Vamos a explicar qué técnicas se acabaron usando y cuales no.

La investigación de las representaciones del tablero nos llevó a tomar una representación de una lista de 128 posiciones de punteros a piezas. Esto se hace para controlar mejor cuando las fichas se salen del tablero. En este caso las piezas no conocen su posición dentro del tablero. Se implementó una matriz de transposición para guardar tableros repetidos. La técnica elegida para implementar el algoritmo de búsqueda fue una poda alfa-beta con búsqueda de la quietud y heurística *killer*. También se ordena la lista de movimientos a explorar en capturas y no capturas. Dentro de capturas se usa el orden MVV/LVA. Se implementó también la profundización iterativa con búsqueda de variación principal. Estos dos procedimientos pueden activarse o desactivarse. Inicialmente están desactivados ya que en las pruebas se comprobó que no funcionaban del todo bien. La profundidad de exploración es de 3 movimientos. En el caso de haya quietud se explora 1 movimiento más y si hay jaque 2 movimientos más.

En el caso de la función de evaluación se siguió el documento de investigación que se creó en su momento. Por supuesto no están todas las características que en dicho documento se citan, podemos decir que la función de evaluación es un resumen de dicho documento.

Se incluyen tres niveles de dificultad (fácil, medio y difícil). Estos niveles tienen distintos niveles de profundidad en el árbol de búsqueda. El fácil explora un nivel, el medio dos y el difícil tres.

8. Pruebas

8.1. Introducción

En este punto vamos a explicar algunas de las pruebas que se realizaron sobre las distintas aplicaciones y los fallos más importantes que rebelaron estas pruebas.

8.2. Programa del profesor

Las pruebas de la aplicación se realizaron durante las 2 últimas semanas antes de la entrega; a la vez que se iba probando se iban corrigiendo los fallos encontrados. Se probaron una a una todas las opciones del menú del programa. Para probarlas todas se generaron ejercicios de todos los capítulos del curso y de todos los tipos de ejercicio disponibles. Se intentaron también generar ejercicios erróneos para comprobar que no dejaba generarlos. Todas las pruebas resultaron satisfactorias y sólo se encontraron fallos en ciertas restricciones de algunos ejercicios que se pudieron solucionar fácilmente.

8.3. Programa del alumno

Igual que con el programa del profesor, las pruebas de este programa se realizaron en las 2 últimas semanas. Las pruebas de la parte que juega al ajedrez se comentan en el punto siguiente. Las pruebas de esta aplicación fueron más engorrosas que las pruebas de la del profesor. Para probarlo se generó con el programa del profesor un curso completo con varios ejercicios por capítulo y ejercicios de todos los tipos. Los ejercicios se fueron probando uno a uno. Varios tipos de ejercicios dieron pequeños fallos que pudieron ser solucionados. Una parte que dio muchos problemas fue la de cargar los ejercicios de los ficheros. Finalmente se solucionó también este problema. Las aperturas son un tipo de ejercicio especial que no crea el profesor. Para probarlas se crearon 19 aperturas distintas. Tanto la creación de usuarios y la gestión de los mismos fueron probadas. También se probaron todas las opciones de la interfaz gráfica. Las pruebas de esta aplicación sacaron a la luz muchos más fallos que las del profesor. También es verdad que este programa es mucho más grande y complejo que el anterior. Decir que al final todos los errores fueron solucionados a tiempo de entregar el proyecto.

8.4. Programa de ajedrez

Las pruebas del programa que jugaba al ajedrez se realizaban a la vez que se iba desarrollando el mismo. Además de pruebas de que no hubiese fallos, el principal objetivo de las pruebas sobre este programa era conseguir alcanzar un nivel de juego aceptable. Para ello, en las primeras etapas probábamos el programa nosotros mismos, jugando partidas. Cuando la aplicación empezó a jugar razonablemente bien, necesitamos alguna herramienta que nos dijese realmente el nivel ELO de nuestro programa. Para ello usamos el programa de ordenador *Chessmaster 7000*. En este programa tienes diversos oponentes a elegir cada uno con una ELO distinta. Así disputando partidas contra el *Chessmaster* pudimos comprobar el nivel de juego de la aplicación. En la aplicación existen 3 niveles de juego vamos a explicar el ELO de cada nivel.

- Nivel fácil: ELO 200 aproximadamente. Utiliza 1 nivel de profundidad, por lo que no ve más movimientos que los suyos, por lo que no tiene en cuenta muchas capturas. Algunas veces parece que hace movimientos aleatorios. En el juego medio compite de sobra con un ELO 250-300 pero es incapaz de dar jaque mate.
- Nivel medio: ELO 700 aprox. Las pruebas comenzaron jugando contra adversarios de nivel 400 y 500 a los que gano con facilidad, contra adversarios de nivel 700 gana ampliamente en el juego medio pero no en los finales. La última partida se jugó con un ELO 815 que ganó. Se observan bastantes mejoras con respecto al nivel anterior en todos los aspectos del juego.

- Nivel difícil: ELO 1100-1200 aprox. Comenzamos jugando contra ELO 950 para ir subiendo progresivamente. Con ELO 950 gana después de una partida larga. Todas las partidas en este nivel suelen ser largas y muy tácticas. Contra un ELO 1100 controla el juego medio y logró dar jaque mate (pero con cierta dificultad). Al jugar con un ELO 1300 perdió la partida. Este es el nivel de juego más alto que alcanza el programa.

Los principales fallos detectados durante las pruebas estaban centrados en la quietud. Costó un par de semanas depurar completamente la aplicación. Otros fallos descubiertos eran ciertos problemas con las claves de la matriz de transposición, problemas en la generación de movimientos, etc. Pero la principal fuente de fallos y que luego se encontró fue la inclusión del enroque y de comer al paso. Esto hizo que la complicación del código del programa creciera mucho con la consiguiente dificultad en su depuración.

9. Valoración

A lo largo del año se nos plantearon una serie de objetivos a ir desarrollando, de los cuales completamos con éxito la mayoría de ellos; algunas tareas fueron propuestas como opcionales, pero en algunos casos decidimos no realizarlas debido a que no teníamos conocimiento de la materia necesaria para llevarlos a cabo, y eso requería dedicar tiempo al aprendizaje de nuevos conceptos; en otros casos pensamos que era mejor concentrar todo nuestro esfuerzo en algunas partes que habíamos hecho, pero que no estaban del todo perfeccionadas, en vez de dedicar nuestro tiempo en aquellas tareas nuevas, dejando algunos apartados no completos del todo.

OBJETIVOS CUMPLIDOS:

- Conseguimos desarrollar con éxito una aplicación que permitiese crear ejercicios cumpliendo todos los requisitos impuestos inicialmente; es decir, se dispone de una interfaz amigable a través de la cual se pueden crear todo tipo de ejercicios de una forma intuitiva.

- Conseguimos desarrollar con éxito una aplicación en la que se podía seguir un curso de ajedrez mediante la realización de ejercicios creados anteriormente por la anterior aplicación. Además se añadió la opción de jugar determinadas aperturas (en total el sistema dispone de 19) con explicaciones breves y concisas de cada uno de los movimientos de éstas.

- Desarrollamos un sistema capaz de jugar partidas de ajedrez contra el usuario con un nivel bastante aceptable; antes de ello se siguió un proceso de investigación donde se consiguió información que luego utilizamos para diseñar dicha aplicación. Además el sistema permite salvar partidas que luego pueden ser continuadas por el usuario en otro momento, y se ofrece la posibilidad de retroceder el último movimiento del usuario en el caso de que no esté de acuerdo.

OBJETIVOS PENDIENTES:

- Inicialmente se dejó en el aire la posibilidad de incluir en el proyecto la opción de guardar el estado del tablero en distintos formatos (pdf, látex, etc). Este objetivo no se realizó debido a que no teníamos conocimientos de látex y pensamos que era mejor emplear nuestro tiempo en mejorar y dejar completamente acabados algunos de los anteriores objetivos.

- En el mes de marzo se nos comentó la posibilidad de añadir una nueva funcionalidad nueva a nuestro proyecto que consistía en permitir jugar partidas de ajedrez por la red. Tras reflexionar sobre esa posibilidad y viendo las tareas que nos quedaban por hacer y el tiempo que teníamos, creímos como mejor opción desechar dicha propuesta y concentrarnos en lo que nos quedaba de proyecto por acabar.

RESULTADO FINAL DEL PROYECTO:

Creemos que hemos cumplido con éxito la mayoría de los objetivos que se nos propusieron en el comienzo del curso, por lo que estamos bastante contentos por como ha quedado el resultado final.

Nuestro principal logro ha sido el haber conseguido un sistema que juegue al ajedrez con un ELO de aproximadamente 1.200; ya que para ello tuvimos que documentarnos durante varios meses y realizar un diseño bastante complejo que nos permitiese jugar contra la máquina consiguiendo tiempos de respuesta de 30 segundos en media.

10. Instalación

Para poder utilizar nuestro proyecto, es necesario tener instalado el *C++ Builder 5*, ó alguna versión más actualizada. Se supone que el usuario dispone de un sistema operativo *Windows 98* ó más actualizado (2000, XP, etc) Ahora se procederá a comentar algunos aspectos de los dos módulos que consta el proyecto.

PROGRAMA DEL CURSO:

Todo el código fuente así como el ejecutable y los directorios *bitmaps* y ejercicios, que serán comentados a continuación, deberán estar en el directorio c:\Ajedrez. Dentro de éste, tendrá que estar el directorio ejercicios, que es donde están todos los ficheros que representan a cada uno de los ejercicios existentes en el curso; también es necesario que se encuentre en c:\Ajedrez el directorio *bitmaps*, el cual contendrá todos los ficheros *bmp* que representan a cada uno de los gráficos que necesita el programa.

PROFESOR:

Todo el código fuente así como el ejecutable y el directorio *bitmaps*, que serán comentados a continuación, deberán estar en el directorio c:\Profesor. Dentro de éste, tendrá que estar el directorio *bitmaps*, el cual contendrá todos los ficheros *bmp* que representan a cada uno de los gráficos que necesita el programa.

11. Bibliografía

Libros:

[1] Ajedrez infantil

[2] Escuela de ajedrez

Antonio Guide

Ediciones Tutor

5ª Edición, 2001

[3] El ajedrez. Curso Completo

Ricardo Aguilera

Ed. Aficiones

[4] Ajedrez infantil

P. Castro, O. Buide, C.Candal, Maria José Castro, J. Feijoo

Ed. Paidotribo

Sitios web:

[5] http://www.xs4all.nl/~verhelst/chess/prograLemming.html

[6] http://www.gamedev.net/reference/articles/article1014.asp

[7]http://www.geocities.com/zodiamoon/amyan/comofuncionan.html

[8] http://members.home.nl/matador/chess840.htm

[9]http://www.brucemo.com/compchess/programming/index.htm

[10]http://www.maths.nott.ac.uk/personal/anw/G13GT1/compch.html

[11] http://www.chessopolis.com/cchess.htm

[12] http://www.ics.uci.edu/~eppstein/180a/s97.html

[13] http://www.chessbrain.net/beowulf/theory.html

[14] http://supertech.lcs.mit.edu/~heinz/dt/

[15] http://www.lkessler.com/index.shtml

Programas consultados:

- *Chessmaster 7000*

12. Comentarios bibliográficos

[Ref5]: En esta página sobretodo hemos sacado información de las representaciones del tablero.

[Ref6]: Un artículo en el que hablan un poco de todo lo relacionado con la programación de ajedrez: representaciones de tablero, técnicas de búsqueda...

[Ref7]: Introducción sencilla a los programas de ajedrez haciendo un comentario de todo lo que necesitan que lleven. Está en español y nos ayudó a realizar la parte de búsqueda.

[Ref8]: Página donde te explican los conceptos generales de la programación en ajedrez aplicados a REBEL (un programa de ajedrez)

[Ref9]: Esta página nos ha sido muy útil para el desarrollo de nuestro proyecto. Tiene muy buena información de todos los temas relacionados con la programación de ajedrez y nos ha sido especialmente útil la parte de la matriz de transposición.

[Ref10]: Página de información general sobre programación de jugos de ajedrez.

[Ref11]: Página de enlaces a otras páginas de ajedrez. Nos ha sido fundamental, ya que la mayoría de las páginas aquí referenciadas las encontramos a través de esta.

[Ref12]: Esta página nos aportó información sobre las representaciones del juego.

[Ref13]: Página usada de referencia para consultar cosas y afianzar conceptos que encontramos en otros sitios y queríamos contrastar.

[Ref14]: Una página que consultamos sobretodo para la obtención de movimientos.

[Ref15]: Una página con muchos enlaces útiles a otras páginas de ajedrez.