Sistemas Informáticos
Curso 2002-03

Herramienta para generar sobre Web ejercicios de ajedrez interactivos para principiantes

Jaime Lamas Legein
Mª Pilar Palomino Palomino
Héctor Sánchez Jean

Dirigido por:

Prof. Cristóbal Pareja Flores
Dpto. Sistemas Informáticos y Programación

Facultad de Informática
Universidad Complutense de Madrid

Índice

0. Resumen en español e inglés

Resumen:

Este proyecto consiste en la realización de una herramienta para generar ejercicios de ajedrez para principiantes. Con dicha herramienta se permiten crear varios tipos de ejercicios con la finalidad de que un alumno con un nivel básico de conocimientos ajedrecísticos pueda practicar lo aprendido. El resultado final de estos ejercicios se visualiza en un Applet lo que permite su integración en una página Web. A tal efecto y para facilitar el aprendizaje se ha creado un sitio Web, donde se explican las nociones básicas del ajedrez en la que se incluyen ejemplos de ejercicios tipos generados con la herramienta desarrollada.

Palabras claves:

Ajedrez, Applet, ejercicios, herramienta, Libro electrónico, piezas, principiante, tablero, Web.

Resumen en inglés:

This project consists on the realization of a tool to generate exercises of chess for beginners. This tool allows creating several types of exercises with the goal that a student with a basic level of chess knowledge can practice his skill. The final result of these exercises is visualized in an Applet that allows it integration on the Web. To facilitate the learning, a Web site has been created. This Web site includes basic knowledge of chess with examples of exercises types generated with the developed tool.

Key words:

Chess, Applet, exercises, tool, interactive book, pieces, beginner, board, Web.

1. Introducción

1.1. Motivación

La idea que se propuso como tema para el proyecto de la asignatura Sistemas Informáticos surge de la posibilidad de favorecer de alguna manera el aprendizaje del juego del ajedrez a los niños. Inicialmente se barajaron dos posibilidades: por un lado desarrollar una interfaz de un libro electrónico de ajedrez, y por otro lado el desarrollo de una herramienta para generar ejercicios de ajedrez. La segunda idea fue la que finalmente escogió el grupo; parecía la más atractiva para adquirir nuevos conocimientos y también por su originalidad, lo que implicaba un esfuerzo extra de investigación y estudio del dominio.

El hecho de facilitar la generación de los ejercicios necesarios para que los más pequeños fueran asimilando los conocimientos, se pensó que era una buena forma de incentivar este aprendizaje, y además, casaba perfectamente con la idea de acercar el ajedrez a los más pequeños. Como se puede ver, lo que se pretendía era dar un soporte adecuado al profesor/tutor para que pudiera elaborar ejercicios de una forma rápida y sencilla y, a la vez, atractiva e intuitiva para el niño.

Siempre partiendo de la idea original, se intentó pensar cómo sería posible ayudar a aquellas personas que quieren enseñar a jugar al ajedrez a los más pequeños y, claro está, usando la informática como medio de apoyo. Está claro que hoy en día la informática y el ajedrez van cogidos de la mano, de ahí que incentivar a los niños en el uso del ordenador para jugar el ajedrez sea lo más lógico. También es cierto que el mercado está lleno de juegos de ajedrez, tanto de aplicaciones autónomas como de juegos en red para jugar con otros usuarios. Pero indiscutiblemente estas aplicaciones no están destinadas al público infantil, y mucho menos si uno se centra en el ámbito del aprendizaje. Por lo tanto, desde este punto de vista se pretende dar repuesta al vacío que existe en la enseñanza del ajedrez al público infantil por medio del ordenador.

Con todo, el proyecto pretende no sólo servir para dar repuesta a la idea original en si misma, sino también en su posible inclusión como módulo en otras aplicaciones en las que se quisiera enseñar de una manera interactiva el juego del ajedrez a los más pequeños.

1.2. Visión general del problema

El proyecto tiene claramente dos puntos de ataque: por un lado se tiene el punto de vista que genera los ejercicios y por otro lado el que los resuelve.

En el primer caso se trata de una persona que quiere generar problemas de ajedrez, es decir, tiene que tener la posibilidad de generar de una forma sencilla e intuitiva ejercicios básicos de ajedrez. Cuando se habla de ejercicios básicos de ajedrez se trata de aquellos ejercicios pensados para que un niño pueda disfrutar aprendiendo con ellos. Estos ejercicios aparecen generalmente en los primeros temas de cualquier libro para aprender a jugar al ajedrez y, sobre todo, en los libros de ajedrez destinados al público infantil. Éstos suelen estar caracterizados por la inclusión de dibujos y por su nivel ajustado.

Desde el punto de vista del que tiene que resolver el problema, que como se ha señalado será normalmente un niño, se tendrá que tener en cuenta las características del mismo a la hora del resultado final del ejercicio generado. Por lo tanto se han de buscar ejercicios vistosos e intuitivos, en los que no se presuponga casi ningún conocimiento en informática.

Como se puede observar, en los dos puntos de vista en los que se ha enfocado el problema prevalece un aspecto clave: facilidad de uso e intuitividad. Esto es así porque el programa pretende satisfacer a un público que no tiene por qué estar muy familiarizado con la informática, y por lo tanto, la complejidad en el manejo tanto de la herramienta que genere los ejercicios como del ejercicio en si ha de ser mínima. Esto a su vez genera dos problemas muy importantes para el transcurso del proyecto:

- La elección de los ejercicios que la herramienta podrá generar.
- El cómo generar un ejercicio a partir de otro programa.

A la hora de pensar en los posibles ejercicios de ajedrez, lo más importante es saber agruparlos en categorías; ya sea teniendo en cuenta el nivel de conocimiento de ajedrez del jugador o bien las características funcionales que implique el ejercicio. El número de ejercicios tipo que la herramienta ha de ser capaz de generar tiene que ser lo suficientemente amplio como para que un niño sin ningún conocimiento previo sea capaz a medida que vaya resolviendo los ejercicios, de, finalmente, jugar una partida completa con los conocimientos básicos del juego.

Por otro lado, la herramienta ha de poder personalizar cada tipo de ejercicio que se considere, de tal forma que cuando se genere el mismo aparezcan reflejados correctamente. Además, se plantea otro problema importante: qué nivel de *inteligencia* tiene que soportar tanto la herramienta como el *programa* ejercicio. Si como se ha comentado al final el niño ha de poder jugar una partida, es indudable que en algún momento será indispensable incorporar *inteligencia*, bien a la herramienta bien al propio programa ejercicio.

1.3. Objetivos

Una vez esbozadas las ideas generales del proyecto, se va a detallar por separado los objetivos que se pretendían alcanzar, y el grado de cumplimiento de cada uno. A saber:

1. Una herramienta/aplicación que de forma visual y sencilla permita crear ejercicios de ajedrez de una categoría determinada y de dificultad creciente. El cumplimiento de este objetivo ha sido casi total, puesto que el número de ejercicios generables es limitado.

2. Cada ejercicio tipo que se desee crear, se tiene que poder personalizar teniendo en cuenta las características del ejercicio y las preferencias del usuario. El cumplimiento de este objetivo ha sido total; el usuario al crear un ejercicio puede introducir los mensajes de error y de acierto que considere oportunos a la hora de generar el ejercicio en la herramienta. Así mismo, debe introducir la pregunta que desea mostrar al alumno a la hora de presentar el ejercicio.

3. Estas características se verán reflejadas, generalmente, por un tablero donde se colocarán las piezas necesarias para resolver el ejercicio, y una serie de mensajes a rellenar que permitirán personalizar el ejercicio. El cumplimiento de dicho objetivo es total. El tablero en el ejercicio final aparece con la misma colocación de piezas que se configuró en la herramienta generadora.

4. Cuando el profesor/tutor esté de acuerdo con el ejercicio que ha creado, podrá generarlo de tal forma que el ejercicio se convierta en una pequeña aplicación independiente. El cumplimiento de

este objetivo es total, ya que cada ejercicio generado se convierte en un Applet totalmente independiente de la herramienta inicial.

5. Esta pequeña aplicación tendrá que ser fiel a las pautas que eligió el profesor/tutor, y tendrá que informar al alumno (generalmente un niño) si ha resuelto correctamente cada ejercicio. El cumplimiento de este objetivo es total, ya que en cada ejercicio se muestra el resultado, o bien un mensaje indicando si la respuesta es correcta o no, cuando el alumno lo desee.

6. Como ejercicio más sofisticado, existirá la posibilidad de jugar una partida completa de ajedrez contra el ordenador, en la que se podrá ponderar el nivel de conocimiento de determinadas estrategias de juego. Este objetivo no se ha logrado totalmente, la calidad de respuesta de la máquina estará limitada por la profundidad de búsqueda en el árbol de juego (2), con lo cual hay movimientos que no corresponden con lo que sería una buena jugada.

7. En un principio, se deseaba que el nivel de juego de la máquina estuviese condicionado directamente por el grado de conocimiento de su oponente, es decir, si el profesor/tutor quiere que el niño practique lo aprendido hasta ese momento por medio de una partida omitiendo conocimientos especiales como el enroque, la promoción del peón o la comida al paso, la máquina tendrá en cuenta estas restricciones para responder con jugadas que estén dentro de los conocimientos del aprendiz. Este aspecto de la herramienta no se ha llevado a la práctica.

8. Inicialmente se pensó crear un tipo específico de ejercicios para el mate en X jugadas, pero finalmente se decidió generarlos del mismo modo que los de jaque, cuidando especialmente la colocación de las piezas en el tablero.

9. En todo momento se tuvo la ambición de que la herramienta funcionase en distintas plataformas, principalmente en Linux y en Windows y que los ejercicios se pudiesen visualizar sobre cualquier navegador apto para esos sistemas operativos. Todo esto se ha cumplido parcialmente, puesto que la herramienta generadora de ejercicios solo se puede ejecutar bajo Windows. Los ejercicios se pueden visualizar en Internet Explorer, Netscape y Mozilla (en el apartado de compatibilidades se concretan las versiones).

Es importarte recordar que todos estos objetivos tienen que sustentarse bajo los principios generales que mueven el proyecto: la sencillez de manejo, que sea interactivo con el usuario y, finalmente, que sea atractivo tanto para el que crea los ejercicios como para el que los resuelva.

1.4. A quién va dirigido

El público al que está destinado esta aplicación se podría describir como aquellos aficionados al ajedrez que buscan una forma de automatizar la generación de ejercicios. Aunque inicialmente está destinado para servir de herramienta de apoyo a la hora de enseñar a jugar a un niño, también es susceptible de ser usado como herramienta para generar ejercicios a cualquier nivel. Por lo tanto, docentes de la materia o sencillamente padres que quieren enseñar a jugar a sus hijos, son colectivos que pueden estar interesados en una herramienta de este tipo.

A pesar de que el ajedrez está muy extendido en el ámbito de la informática e Internet, es difícil encontrar cursos en los que se pueda practicar de forma interactiva el conocimiento que se va adquiriendo. Sin duda, una herramienta de este tipo puede favorecer a aquellas personas encargadas de dar cursos de ajedrez (aficionados o profesionales), a darle un cariz diferente a sus cursos, pues al automatizar la generación de ejercicios se abandonaría el carácter estático de los que se proponen a la hora de aprender a jugar al ajedrez, más aún cuando se extrapola al público infantil.

Por consiguiente, se pueden resumir en tres perfiles los posibles candidatos que pueden estar interesados en la aplicación. A saber:

1. Personas encargadas de dar cursos de Ajedrez.
2. Padres que quieren enseñar a jugar a sus hijos.
3. Simples aficionados que por alguna razón necesiten generar ejercicios de Ajedrez.

1.5. Posibles aplicaciones actuales y futuras

Como se ha comentado anteriormente la aplicación principal del proyecto es la de servir como herramienta de apoyo a la hora de crear ejercicios de ajedrez por ordenador. Pero no hay que perder de vista el hecho

de que hoy en día las tecnologías y diferentes plataformas que hay en el mercado tienen cada vez más facilidades a la hora de comunicarse entre si. Esto se traduce en que muchas de las aplicaciones y herramientas que se utilizan en un equipo electrónico (por ejemplo un ordenador) sean relativamente fáciles de adaptar a otras plataformas.

En este caso particular, esto es equivalente a decir que, si en un principio esta herramienta está destinada a aprovechar las facilidades que ofrece un ordenador, no es descabellado pensar que esta misma herramienta se pudiera implementar en otras plataformas (véanse teléfonos móviles o televisión digital). De hecho esta circunstancia se está dando habitualmente con muchas aplicaciones que existían en la informática. Para ser más claros, se podría pensar en los cursos interactivos que ya existen en la televisión digital, en los que se puede interactuar con la información que se ofrece por medio de dispositivos similares a los que hay en la informática.

De igual manera ocurre con los teléfonos móviles que ya utilizan tecnologías de programación compatibles con los que se emplean habitualmente en la informática. La oferta que ofrece la telefonía es si cabe más grande, pues no sólo se puede pensar en cursos que se pudiera “bajar” el usuario (con sus consiguientes ejercicios generados con la herramienta) sino en que un profesor de ajedrez o cualquier aficionado en la materia pudiera mandar un ejercicio a un alumno o a varios o a otro aficionado punto a punto. Con esta herramienta se conseguirían resultados más que aceptables empleando poco tiempo y esfuerzo.

Por lo tanto, bien como herramienta autónoma, bien participando conjuntamente en una aplicación más grande, se pueden sacar varias utilidades - y en varias plataformas diferentes - a la aplicación que trata el proyecto.

En cualquier caso, no es el objetivo actual del proyecto llegar a tal profundidad en el desarrollo de la herramienta. Simplemente se ha intentando dar una idea de los posibles enfoques más avanzados que puede ofrecer este proyecto, dejando la puerta abierta a otras posibles relaciones que se podrían establecer con otros programas.

2. Descripción general

2.1. Antecedentes del proyecto

Desde los inicios de la informática siempre se ha prestado especial interés en todo aquello en lo que *la máquina* pudiera enfrentarse a la inteligencia humana. Como es natural, el ajedrez ha sido uno de los juegos donde más se ha investigado por su complejidad inherente, y más si cabe según ha ido aumentando la capacidad de cálculo de los ordenadores. Con la escalada exponencial en cuanto a las prestaciones que se lleva produciendo de forma continuada en los últimos años, han surgido muchos proyectos que con mayor o menor éxito han intentado vencer a los actuales maestros del juego del ajedrez. Lo cierto es que este aumento de prestaciones en un principio sólo se enfocó al desarrollo de programas destinados al enfrentamiento directo con humanos o bien con otras máquinas, siendo habitual desde hace ya muchos años, certámenes para determinar los programas más potentes y expertos de la materia. Pero no ha sido hasta la revolución multimedia y más aún, con la explosión de Internet, cuando la informática ha penetrado en otros campos siendo la enseñanza por ordenador uno de ellos.

Dentro del mundo del ajedrez, el esfuerzo más importante ha seguido destinado a la creación de aplicaciones para que la máquina juegue contra el humano. Este punto de vista se ha desarrollado en dos planos: aplicaciones profesionales y aplicaciones lúdicas. Mientras que en el primero únicamente se ha buscado la calidad de juego de la máquina, en el segundo se ha descuidado más este aspecto y se ha incidido en el aspecto visual y facilidad de uso. Esto ha desembocado en infinidad de títulos que se pueden adquirir en el mercado. Centrándose en el plano didáctico, Internet ha sido sin duda la gran protagonista; se ha establecido toda una comunidad en Internet en la que la forma más habitual de jugar ha pasado a ser a través de la red. Hoy en día, hay infinidad de sitios donde se puede participar en partidas con personas de todo el mundo. Esto ha llevado a la creación de innumerables programas que posibilitan a dos personas jugar por red. El carácter didáctico de este punto es obvio, pues la posibilidad de jugar con cualquier persona hace que se aprenda mucho más que si siempre se juega con la misma. Sin abandonar Internet, también se ha utilizado este medio como fuente de datos de partidas. De hecho, se han creado varios estándares (como el PGN siglas de *Portable Game Notation*) para la notación de las partidas, limitándose su uso a la representación de jugadas en tableros.

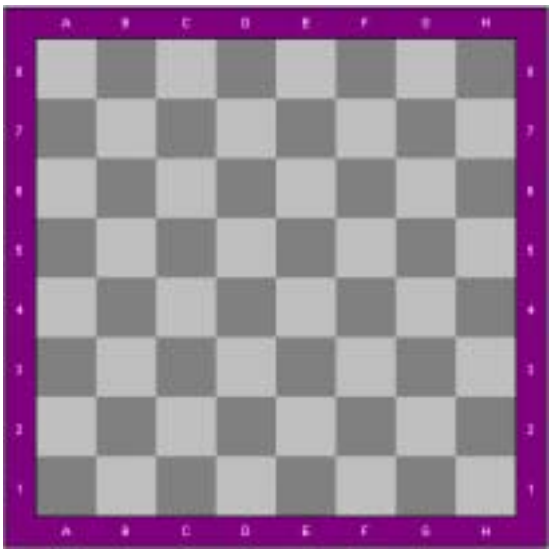
No obstante, en lo que se refiere al aprendizaje infantil del ajedrez, no se ha desarrollado tanto como cabría esperar. Más aún, conscientes de que para la mayoría de las ciencias y aprendizajes básicos infantiles, existen programas que sirven de apoyo. Pero lo cierto es que el ajedrez no ha tenido tanto calado debido a que siempre se ha asociado a personas adultas e incluso muchos adultos son reticentes a aprender el juego por el aparente esfuerzo que conlleva.

Aunque no se encuentre exactamente en el mismo ámbito que este proyecto, existen algunas páginas de Internet en las que se enseñan nociones básicas del juego, y en las que quizás un niño pudiera sacar algún partido. En todo caso, siempre se trata de páginas en las que sólo se incluyen textos, y en las que los ejercicios que se plantean son del todo estáticos. También es cierto que algunos programas de ajedrez encuadrados en el segmento lúdico, tienen apartados destinados al aprendizaje de los aspectos básicos del juego. Lo que sucede es que al no ser el fin mismo del programa, se suelen descuidar muchos aspectos.

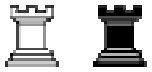
Lo que está claro es que una herramienta destinada a facilitar la generación de ejercicios es, en si misma, una idea bastante novedosa y, como ya se ha comentado, aplicable en distintos frentes que puede abrir el mundo de la informática a la enseñanza infantil del ajedrez.

2.2. Objetivos y funciones

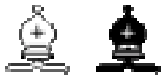
En el juego se emplea un tablero cuadrado de 8 x 8 casillas similar al que se muestra a continuación:



En este tablero la mitad de las casillas son blancas y la otra mitad son negras. A parte del tablero también se usan 32 piezas, de las cuales 16 son blancas y 16 son negras. Para cada color se cuenta con:

2 torres 

2 caballos 

2 alfiles 

1 dama 

1 rey 

8 peones 

El objetivo del juego es ganar una partida al contrario cuando se trata de jugar una partida completa, o bien resolver un ejercicio propuesto con éxito.

A continuación se van a desglosar las funcionalidades generales que tiene que integrar el proyecto. Éstas se van a agrupar en dos categorías: la herramienta que genera el ejercicio y el *programa* ejercicio.

Herramienta generadora de ejercicios.

- Conjunto de *cuadros de diálogo* encadenados en los que a través de preguntas se pueda elegir el tipo de ejercicio que se desea crear.

- Una vez conocido el tipo de ejercicio que se quiere realizar se generará un tablero con todas las piezas disponibles en el juego del ajedrez. Éstas se deben poder arrastrar y soltar en el tablero para formar una configuración.

- Según el tipo de ejercicio seleccionado, se deberá comprobar que realmente se puede determinar una solución al ejercicio planteado con la configuración del tablero en aquellos tipos en los se requiera.

- Se ha de dar la posibilidad de personalizar el ejercicio por medio de cajas de diálogo en las que se pueda escribir texto. Especial importancia tendrán los mensajes de error y de éxito que aparecerán en el ejercicio y que también se han de poder incluir por medio de formularios.

- Una vez se haya configurado el ejercicio según las preferencias del usuario, éste ha de poder ser guardado.

- A su vez, y como es lógico, existirá la opción de generar ejercicio, en la que se le darán los datos relevantes acerca del *programa* ejercicio generado.

- Una vez la herramienta disponga de todos los datos del ejercicio, generará un *programa ejercicio* en el que se reflejará adecuadamente la configuración dada por el usuario. Por lo tanto, existirá una estructura de datos que guarde y comunique al *programa ejercicio* la configuración establecida por el usuario a través de la herramienta.

Desde el punto de vista del programa ejercicio:

- Debe aparecer un tablero de ajedrez con las piezas colocadas según el usuario indicó en la herramienta que genera el ejercicio.

- Se mostrará un texto (según tipo de ejercicio) en el que se explicará al *alumno* lo que ha de conseguir y las instrucciones que debe seguir.

- En todo momento se le ha de mostrar por medio de mensajes el estado en el que se encuentra el ejercicio, haciendo especial hincapié en el caso de llegar a una solución correcta o incorrecta. En este último caso se le informará de cuál era la respuesta correcta.

- Cuando sea necesario, las piezas se podrán mover siguiendo fielmente las reglas establecidas en el ajedrez. Se informará en la medida de lo posible de todo aquello que no siga las reglas del ajedrez.

- Se tendrá especial cuidado en el aspecto visual de todos los elementos, teniendo siempre presente que está destinado al público infantil.

3. Requisitos específicos

En este apartado se van a clasificar y explicar cada uno de los requisitos identificados en el proyecto.

3.1. Requisitos funcionales

Se van a explicar por un lado los requisitos correspondientes a la herramienta que genera los ejercicios y por otro lado los requisitos correspondientes a los ejercicios generados.

3.1.1. Requisitos de la herramienta/ejercicio

Funciones principales:

Función InicializarTablero: Crea un tablero de ajedrez sin ninguna pieza

Entrada:

Salida: Tablero de ajedrez

Usa: Tablero

Actualiza: Tablero de ajedrez

Efecto: Se crea un tablero de ajedrez vacío, preparado para que el usuario pueda poner los ejercicios deseados.

Excepciones:

Función ColocarPiezas: Se muestran las piezas y un tablero vacío y el usuario ha de colocar sobre el tablero las piezas que considere oportunas

Entrada: Tablero vacío, todas las piezas de ajedrez

Salida: Tablero con la configuración inicial para un determinado ejercicio elegido por el usuario.

Usa: Tablero, piezas

Actualiza: Tablero de ajedrez

Efecto: Tablero con piezas.

Excepciones: Si no se encuentran las piezas se informa de ello.

Función GuardarConfiguraciónInicialTablero: Se guarda la configuración del tablero que se ha obtenido al colocar las piezas en un tablero inicial

Entrada: Tablero con una configuración inicial

Salida: Fichero con la configuración inicial del tablero

Usa: Tablero de ajedrez y base de datos de configuraciones de tableros

Actualiza: Base de datos de configuraciones de tableros

Efecto: Se añade un nuevo fichero a la base de datos que almacena todas las configuraciones de tableros que se tienen

Excepciones: Ya existe un fichero con ese nombre, la herramienta avisa de ello, o no hay suficiente espacio para guardar una configuración inicial nueva, la herramienta avisa de ello

Función GuardarMensajes: Guarda un fichero con los mensajes que se han de mostrar al alumno al desarrollar un tipo de ejercicio determinado

Entrada: Mensajes escritos por el usuario

Salida: Fichero con los mensajes escritos por el usuario

Usa: Mensaje escrito por el usuario y la base de datos

Actualiza: Base de datos de mensajes (al alumno)

Efecto: Se añade un nuevo fichero a la base de datos que almacena todos los mensajes que se le han de mostrar al alumno cuando desarrolle un ejercicio determinado

Excepciones: No hay suficiente espacio en la base de datos para almacenar un nuevo fichero, la herramienta avisa de ello

Función RecuperarConfiguraciónTablero: Recupera una configuración que haya guardada en la base de datos de configuraciones de tableros

Entrada: Nombre del fichero que se quiere recuperar

Salida: Tablero con la configuración recuperada

Usa: Base de datos de configuraciones de tableros

Actualiza: Tablero

Efecto: Obtiene una configuración que había sido guarda previamente

Excepciones: No existe el fichero que se quiere recuperar, la herramienta avisa de ello

Función GuardarConfiguraciónHTML: Se genera definitivamente en forma de Applet el ejercicio generado por el usuario

Entrada: Configuración inicial del tablero, mensajes para mostrar al alumno

Salida: Applet (Página HTML, lista para jugar)

Usa: Base de datos de configuraciones de tableros y base de datos de mensajes, base de datos con los Applets generados

Actualiza: base de datos con los Applets generados

Efecto: generación automática del ejercicio propuesto por el usuario

Excepciones: Falta algún fichero bien el de la configuración inicial del tablero o el de los mensajes. No hay espacio para guardar un nuevo Applet. La herramienta avisa de todo ello

3.1.2. Requisitos ejercicios generados

Nótese que en este apartado cuando se habla de jugador o usuario se refiere al alumno.

En primer lugar se describen las funciones comunes para todos los ejercicios:

Función CrearCasilla: En esta función se crean todas las casillas del tablero, que se pueden describir como cuadrados blancos y negros

Entrada: Información para saber si se trata de una casilla blanca o negra

Salida: Casilla blanca o negra creada

Usa: Cuadrado

Actualiza: Tablero

Efecto: Es lo más básico, para crea cualquier ejercicio se necesita primero dibujar todas las casillas del tablero

Excepciones:

Función CrearPiezas: En esta función se crean todas las piezas necesarias para jugar al ajedrez

Entrada: Identificación del nombre y color de cada pieza

Salida: Pieza lista para ser colocada en el tablero

Usa:

Actualiza:

Efecto: Es una función básica necesaria para cualquier tipo de ejercicio

Excepciones: Si es imposible cargar las imágenes de las piezas, se muestra un mensaje informando de ello

Función ColocarPiezas: En esta función se colocan las piezas en el tablero según una configuración dada

Entrada: Configuración inicial de las piezas, el tablero y las piezas

Salida: Tablero con una configuración inicial determinada, listo para que el alumno interactúen en él.

Usa: Tablero, Piezas, fichero con la configuración inicial del tablero

Actualiza: Tablero

Efecto: Muestra un tablero con una configuración inicial

Excepciones: Si es imposible leer el fichero con la configuración inicial del tablero, se muestra un mensaje informando de ello

Función ValidarMovimiento: Para cada movimiento que se hace de una pieza dentro del tablero hay que comprobar si es válido teniendo en cuenta de la pieza de la que se trata y de la configuración del tablero sobre el que se mueve dicha pieza

Entrada: Pieza que tiene asociada su posición inicial y final

Salida: Valor que indica si el movimiento para esa pieza es correcto

Usa: Tablero, Pieza, Validación

Actualiza:

Efecto: Da información de si una determinada pieza que se encuentra en una casilla origen, se puede mover a otra casilla destino

Excepciones:

Función MouseDown: Captura los eventos del ratón sobre cualquier casilla del tablero, pieza o sobre un botón del ejercicio al hacer un clic

Entrada: Clic del ratón

Salida: Efecto dependiendo del elemento sobre el que se haga el clic del ratón

Usa:

Actualiza: El ejercicio en general (depende del elemento que capture el clic)

Efecto: Cambio del ejercicio

Excepciones: Si no se puede realizar el MouseDown sobre un elemento, no se hará nada al actuar sobre él

Función MouseUp: Captura los eventos del ratón sobre cualquier casilla del tablero, pieza o botón del ejercicio al soltar el clic del ratón

Entrada: la acción de soltar el clic del ratón

Salida: Efecto dependiendo del elemento sobre el que se haga el clic del ratón

Usa:

Actualiza: El ejercicio en general (depende del elemento que capture el clic)

Efecto: Cambio del ejercicio

Excepciones: Si no se puede realizar el MouseUp sobre un elemento, no se hará nada al actuar sobre él

Función MouseMove: Al desplazarse sobre el tablero se indica (con un dibujo de una mano) qué elementos están activos (sobre cuales se puede hacer un clic)

Entrada: Tablero

Salida: Mano como imagen del cursor del ratón

Usa: Tablero

Actualiza: El cursor del ratón

Efecto: Cambio de la imagen del cursor

Excepciones:

Posteriormente se explican las funciones clasificándolas para cada tipo de ejercicio, y explicando en qué consiste cada ejercicio, para ir familiarizándose con ellos. Las funciones comunes no se vuelven a explicar simplemente se nombran en cada ejercicio aquellas de las que hace uso:

Ejercicio 1. Movimiento básico de piezas.

En este ejercicio se muestra un tablero con una o varias piezas colocadas en determinadas casillas y lo que hay que hacer es marcar las casillas a las que esas piezas se deben mover.

1. El juego se inicia cuando el jugador marca la primera casilla.
2. En cada paso del juego, el jugador debe marcar una casilla, o desmarcar una casilla que hubiese marcado anteriormente o elegir mostrar respuesta.
3. Cuando el jugador elija esta opción se mostrarán las opciones correctas y un mensaje indicándole al usuario si lo ha hecho bien o debe seguir practicando.
4. El jugador en cualquier momento podrá finalizar el juego.

Funciones: CrearCasilla, CrearPiezas, ColocarPiezas, MouseDown, MouseUp, MouseMove.

Función Marcar: Esta función pinta un aspa que se usa cuando el usuario hace un clic sobre una casilla para marcarla (poner un aspa)

Entrada: Tablero con una pieza colocada en él

Salida: Tablero con la pieza en el mismo lugar y una serie de casillas marcadas

Usa: Tablero, Piezas

Actualiza: Tablero

Efecto: Marcar casillas

Excepciones: Si es imposible marcar alguna casilla porque no esté activa no se dibuja la marca.

Función Desmarcar: Esta función quita un aspa que previamente el usuario había puesto sobre una casilla, al hacer un clic sobre ésta

Entrada: Tablero con una pieza colocada en él y alguna casilla marcada

Salida: Tablero con la pieza en el mismo lugar y una casilla menos marcada

Usa: Tablero, Piezas

Actualiza: Tablero

Efecto: Desmarcar casillas

Excepciones: Sólo se podrá desmarcar una casilla que previamente haya sido marcada

Función Pintar solución: Esta función pinta un círculo rojo sobre las casillas solución de un determinado ejercicio. Esto ocurre cuando se elige la opción de mostrar solución

Entrada: Tablero con una pieza colocada en él

Salida: Tablero con la pieza en el mismo lugar y las casillas solución del ejercicio marcadas con círculos rojos

Usa: Tablero, Piezas, Fichero con la solución del ejercicio

Actualiza: Tablero

Efecto: Mostrar la solución del ejercicio

Excepciones: Imposible leer el archivo que contiene la solución, la herramienta informa de ello

Ejercicio 2. Marcar la respuesta correcta

En ese ejercicio se muestra un tablero con una determinada configuración inicial, una pregunta y unas posibles respuestas, de entre las que el jugador ha de marcar la correcta.

1. Las posibles opciones del usuario en cada paso del juego son marcar una opción, desmarcar una casilla que había marcado previamente, o elegir ver la respuesta correcta.
2. Cuando el jugador elija la opción de ver la respuesta correcta se mostrará la opción correcta.
3. El jugador en cualquier momento podrá finalizar el juego.

Funciones: CrearCasilla, CrearPiezas, ColocarPiezas, MouseDown, MouseUp, MouseMove.

Función MostrarSolución: Esta función muestra un mensaje con la solución correcta de la pregunta formulada

Entrada: Tablero con una pieza colocada en él, una pregunta y posibles respuestas

Salida: Solución correcta a la pregunta

Usa: Tablero, Piezas, Fichero de pregunta, Fichero de respuestas, Fichero de solución

Actualiza: ejercicio

Efecto: Mostrar la solución del ejercicio

Excepciones: Imposible leer el archivo que contiene la pregunta, las posibles respuestas o la solución, la herramienta informa de ello

Ejercicio 3. Mover una pieza por un campo de minas

En este ejercicio se muestra un tablero con Piolín en una posición del tablero, una pieza en otra posición del tablero, y una serie de minas salpicadas por todo el tablero. El juego consiste en ir moviendo la pieza, de acuerdo con su movimiento determinado, por el tablero hasta llegar a la posición donde está Piolín, todo esto ha de hacerlo en el menor número de movimientos posibles.

1. El juego se iniciará al hacer el primer movimiento de la pieza correspondiente.
2. Al iniciarse el juego se pondrá el contador de movimientos en marcha.
3. El contador se actualizará en cada movimiento
4. En cada paso del juego, el jugador sólo puede avanzar a otra casilla.
5. Cuando el aprendiz alcance la meta se le dará un mensaje de enhorabuena indicándole el número de movimientos que ha usado para alcanzar a Piolín.
6. La situación de las minas en el tablero es la que el "profesor" puso previamente.

Funciones: CrearCasilla, CrearPiezas, ColocarPiezas, MouseDown, MouseUp, MouseMove.

Función PintarMina: Esta función pinta una mina sobre unas casillas especificadas en un fichero

Entrada: Tablero, imagen de las minas, fichero con las posiciones donde van situadas las minas

Salida: Tablero con las minas situadas en él en las posiciones indicadas en el fichero correspondiente

Usa: Tablero, imagen minas, fichero

Actualiza: Tablero

Efecto: Dibujar minas en un tablero

Excepciones: Imposible leer el archivo que contiene las posiciones de las minas, la herramienta informa de ello

Función PintarMeta: Esta función pinta la imagen de Piolín sobre una casilla especificada en un fichero

Entrada: Tablero, imagen de Piolín, fichero con la posición donde va situada la meta

Salida: Tablero con la meta situada

Usa: Tablero, imagen Piolín, fichero

Actualiza: Tablero

Efecto: Dibujar la meta (Piolín) en un tablero

Excepciones: Imposible leer el archivo que contiene la posición de la meta, la herramienta informa de ello

Ejercicio 4. Jugar una partida de ajedrez

En este ejercicio como es de esperar se jugará una partida de ajedrez completa.

1. El juego se iniciará cuando las blancas (usuario) muevan una pieza
2. En cada paso del juego, el usuario puede realizar el movimiento de una pieza.
3. Cuando el usuario mueve una pieza ha de esperar la respuesta del jugador contrario (máquina), cuando el jugador contrario ya haya respondido ya puede continuar el usuario como le corresponda.
4. Cuando el jugador contrario te da un jaque mate (ataque al rey sin solución para salvarlo) has perdido.
5. El jugador consigue su objetivo cuando le ha dado jaque mate al contrario o en caso de q eso sea posible pues cuando se consiguen tablas.
6. El jugador en cualquier momento podrá elegir si jugar a ganar material o cambiar la importancia que él le dé a los movimientos de cada pieza.
7. El jugador en cualquier momento podrá abandonar la partida.

Funciones: CrearCasilla, CrearPiezas, ColocarPiezas, MouseDown, MouseUp, MouseMove.

Función EstimarJugada: Esta función es el motor de la inteligencia del juego, es aquí donde se valoran todas las jugadas, se hacen estimaciones para cada pieza asignándole a cada una un porcentaje (que indica la importancia de esa pieza en el juego), esos porcentajes varían si se está jugando a ganar material o a conquistar el centro del tablero, en cuyo caso se pasarán a valorar más dichos factores. Esta función se usa cuando se quiere jugar pensando. Otra posibilidad sería jugar de forma aleatoria, donde lógicamente no se hace uso de ningún tipo de "inteligencia"

Entrada: Tablero con unas piezas colocadas en él

Salida: Valor que indica la estimación de ese tablero

Usa: Tablero, Piezas

Actualiza: El valor de estimación

Efecto: Valora las piezas y posiciones de las piezas en un tablero

Excepciones:

Ejercicio 5. Dar jaque en X jugadas

Este ejercicio consiste en dada una configuración inicial del tablero con piezas de ambos colores, en que las piezas blancas (las del usuario) han de dar jaque en X jugadas a las piezas negras (las del contrario)

1. El número de jugadas empieza a contar en el momento que el jugador (blanco) mueve una pieza.
2. En cada paso (uno solo si X es 1 movimiento) el jugador podrá mover una de sus piezas para conseguir el objetivo perseguido.
3. Cuando el jugador agote el número de movimientos que le son permitidos, se le mostrará un mensaje indicándole al usuario si lo ha hecho bien –el rey negro está finalmente en jaque – o debe seguir practicando.
4. El jugador en cualquier momento podrá abandonar el ejercicio.

Funciones: CrearCasilla, CrearPiezas, ColocarPiezas, MouseDown, MouseUp, MouseMove.

Función InicializarContador: Esta función pone el contador (que representa el número de movimientos) a cero

Entrada: Contador

Salida: Contador

Usa:

Actualiza: El valor del contador

Efecto: Inicializa el valor del contador a cero para comenzar a contar el número de movimientos

Excepciones:

Función IncrementarContador: Esta función incrementa el contador (número de movimientos) en una unidad

Entrada: Contador

Salida: Contador

Usa:

Actualiza: El valor del contador

Efecto: Incrementa el valor del contador en una unidad para ver si es posible dar jaque en X jugada.

Excepciones:

La calidad del movimiento que realiza el ordenador en este tipo de ejercicios se podría mejorar aumentando el nivel de profundidad del árbol de juego.

3.2. Requisitos de capacidad

Para poder ejecutar correctamente la aplicación se necesita un disco duro con un espacio libre mínimo de 6 MB. Si bien es cierto que se recomienda el doble de espacio para poder almacenar sin problemas los ejercicios generados.

Los tiempos de respuesta más significativos en la aplicación, para un ordenador Pentium III a 1.5 GHz con un disco duro de 60 Gigas, y 128 de memoria RAM son:

- Cargar un Applet <= 5 segundos.
- Generar los Applets <= 0.1 segundo.
- Mover una pieza como respuesta a un movimiento del usuario <= 4 segundos.
- Mostrar la solución correcta <= 0.1 segundo.

3.3. Requisitos de interfase

Para la presentación de esta herramienta es necesario disponer de un ordenador que tenga instalado algún navegador, necesario para visualizar los Applets (ejercicios).

La interfaz que se usa es sencilla y amigable para que el usuario, tanto el profesor como el alumno, se pueda manejar fácilmente

3.4. Requisitos de operación

Además de los indicados en los subapartados de objetivos y funciones del apartado de descripción general se deberán cumplir los siguientes:

- Para mover una pieza de una casilla a otra hay que arrastrarla con el ratón. Inicialmente se pensó mostrar la pieza en el transcurso del movimiento aunque al final se desechó la idea porque estéticamente no gustó el resultado.
- En el caso de ejercicios de marcar casillas u opciones para marcar una casilla o una opción hay que hacer un clic de ratón sobre ella y para desmarcar una previamente marcada volver a hacer otro clic.
- Para finalizar el juego o un ejercicio bastará con pulsar la tecla de finalización una sola vez.

3.5. Requisitos de recursos

Este sistema podrá ser ejecutado en cualquier PC compatible con la siguiente configuración mínima:

CPU: 350 MHz

Memoria: 64 Megabytes.

Sistema Operativo: Windows 9X o superior.

Navegadores: Internet Explorer versión 5.5 o superior, Netscape versión 6.0 superior, Mozilla versión 5.0 o superior.

Para poder visualizar los Applets es necesario tener instalada la máquina virtual de java.

Aunque debido a la carga de memoria a la hora de jugar una partida completa, se recomienda un ordenador con las siguientes características:

CPU: 1.5 GHz

Memoria 128 Megabytes.

Sistema Operativo y Navegador indiferente dentro de las versiones permitidas.

3.6. Requisitos de pruebas de aceptación

- Permitir la sencillez y fácil manejo para que los usuarios puedan usarlo de forma cómoda.
- Permitir jugar una partida de ajedrez con cierta inteligencia.

3.7. Requisitos de calidad

Se cumplen parcialmente los objetivos especificados. Sería deseable más inteligencia para poder jugar al ajedrez de forma más profesional, haciendo un estudio más detallado de los posibles movimientos y sus efectos en la partida. No obstante la aplicación no se ha centrado en la calidad de juego, puesto que está diseñada para un público infantil.

3.8. Requisitos estructurales

En un mismo PC pueden estar ambas partes, la de la herramienta que genera los ejercicios y los ejercicios generados por la herramienta. De todas formas existen ejemplos de ejercicios generados por la herramienta, que están colgados en la Web (www.iespana.es/ajedrezsi). De modo que cualquier persona puede hacer uso de esos ejercicios y comprobar el resultado de la aplicación.

4. Casos de Uso

1. Caso de Uso: Generar ejercicio

1.1. Sesión: Principal

1.1.1. Caso de Uso: Generar ejercicio

1.1.2. Actores: Usuario (Profesor)

1.1.3. Propósito: Permitir a un usuario (profesor) desarrollar ejercicios de ajedrez.

1.1.4. Resumen: El usuario propone un determinado tipo de ejercicio, establece una configuración inicial del tablero, y los mensajes que posteriormente se mostrarán al jugador.

1.1.5. Tipo: Primario y esencial

1.1.6. Referencias cruzadas:

1.1.6.1. Funciones: InicializarTablero, ColocarPiezas, GuardarConfiguraciónInicialTablero, GuardarMensajes, RecuperarConfiguraciónTablero, GuardarConfiguraciónHTML, ComprobarJaque, ComprobarMate.

1.1.6.2. Casos de uso: Ejercicio1, Ejercicio2, Ejercicio3, Ejercicio4, Ejercicio5.

1.1.7. Curso normal de los eventos:

	Acción de los actores		Respuesta del sistema
		1.	Se despliega una ventana que solicita el tipo de ejercicio que el usuario quiere realizar
2.	El usuario elige entre: a. Marcar los posibles movimientos de una pieza b. Hacer una pregunta a partir de una configuración de tablero ofreciendo varias posibilidades c. Atravesar un campo de minas con una figura d. Jugar una partida completa activando opciones e. Dar jaque en X jugadas		
		3.	Cierra la ventana que desplegó

1.2. Sesión: Marcar los posibles movimientos de una pieza

1.2.1. Curso normal de eventos:

Acción de los actores	Respuesta del sistema
1. El usuario eligió la opción de marcar los posibles movimientos de una pieza	
	2. El sistema muestra un tablero vacío y todas las piezas del ajedrez
3. El usuario puede elegir entre las siguientes acciones: a. Colocar una pieza en el tablero. b. Recuperar una configuración guardada previamente.	

1.2.1.1. Sesión: Colocar una pieza en el tablero:

1.2.1.1.1. Curso normal de eventos:

Acción de los actores	Respuesta del sistema
1. El usuario eligió la opción de colocar una pieza en el tablero. Coloca una pieza.	
	2. El sistema le da la opción de guardar la configuración inicial del tablero con un nombre.
3. El usuario elige el nombre con el que quiere guardar dicha configuración	
	4. El sistema genera un archivo de texto con la configuración inicial guardada.

1.2.1.1.2. Cursos alternos:

Línea 4: Imposible guardar la configuración en la base de datos de configuraciones, se muestra un mensaje de error.

1.2.1.2 Sesión: Recuperar una configuración guardada:

1.2.1.2.1. Curso normal de eventos:

Acción de los actores	Respuesta del sistema
1. El usuario eligió la opción de recuperar una configuración inicial guardada	
	2. El sistema solicita el nombre del archivo que contiene la configuración inicial guardada
3. El usuario escribe el nombre del fichero que quiere recuperar.	
	4. El sistema actualiza automáticamente el tablero con la configuración que ha recuperado de ese archivo de texto.

1.2.1.2.2. Cursos alternos:

Línea 3: Imposible recuperar la configuración de la base de datos de configuraciones, bien porque no existe o porque no permite acceder a esa copia, se muestra un mensaje de error.

1.3. Sesión: Hacer una pregunta a partir de una configuración de tablero ofreciendo varias posibilidades

1.3.1 Curso normal de eventos:

Acción de los actores	Respuesta del sistema
1. El usuario eligió la opción de hacer una pregunta con una configuración de tablero ofreciendo varias posibilidades	
	2. El sistema muestra un tablero vacío y todas las piezas del ajedrez
3. El usuario puede elegir entre las siguientes acciones: a. Colocar una pieza en el	

tablero.

b. Recuperar una configuración guardada previamente.

1.3.1.1. Sesión: Colocar una pieza en el tablero:

1.3.1.1.1. Curso normal de eventos:

Acción de los actores	Respuesta del sistema
1. El usuario eligió la opción de colocar una pieza en el tablero. Coloca una pieza.	
	2. El sistema le da la opción de guardar la configuración inicial del tablero con un nombre.
3. El usuario elige el nombre con el que quiere guardar dicha configuración	
	4. El sistema genera un archivo de texto con la configuración inicial guardada. Muestra al usuario la configuración establecida y solicita la pregunta a hacerle al alumno, las posibles respuestas y la respuesta correcta.
5. El usuario introduce todos los datos solicitados por el sistema	
	6. El sistema guarda tres ficheros uno con la pregunta, otro con las posibles respuestas y otro con la respuesta correcta. Finalmente solicita al usuario un nombre para ejercicio generado.
7. El usuario introduce el nombre para ese ejercicio.	
	8. El sistema genera el ejercicio con las opciones deseada por el profesor

1.3.1.1.2. Cursos alternos:

Línea 4: Imposible guardar la configuración en la base de datos de configuraciones, se muestra un mensaje de error.

Línea 6: Imposible guardar los tres ficheros si hay algún dato que el usuario no ha introducido, se muestra un mensaje indicando que debe introducir todos los datos.

Línea 7: Si el usuario no introduce el nombre del fichero, el sistema da un mensaje que informa de ello.

Línea 8: Imposible guardar el ejercicio por falta de espacio u otra causa, el sistema informa del error.

1.3.1.2 Sesión: Recuperar una configuración guardada:

1.3.1.2.1. Curso normal de eventos:

Acción de los actores	Respuesta del sistema
1. El usuario eligió la opción de recuperar una configuración inicial guardada	
	2. El sistema solicita el nombre del archivo que contiene la configuración inicial guardada
3. El usuario escribe el nombre del fichero que quiere recuperar.	
	4. El sistema actualiza automáticamente el tablero con la configuración que ha recuperado de ese archivo de texto.
5. El usuario introduce todos los datos solicitados por el sistema	
	6. El sistema guarda tres ficheros uno con la pregunta, otro con las posibles respuestas y otro con la respuesta correcta. Finalmente solicita al usuario un nombre para ejercicio generado.
7. El usuario introduce el nombre para ese ejercicio.	
	8. El sistema genera el ejercicio con

las opciones deseada por el profesor

1.3.1.2.2. Cursos alternos:

Línea 3: Imposible recuperar la configuración de la base de datos de configuraciones, bien porque no existe o porque no deja acceder a esa copia, se muestra un mensaje de error.

Línea 6: Imposible guardar los tres ficheros si hay algún dato que el usuario no ha introducido, se muestra un mensaje indicando que debe introducir todos los datos.

Línea 7: Si el usuario no introduce el nombre del fichero, el sistema da un mensaje que informa de ello.

Línea 8: Imposible guardar el ejercicio por falta de espacio u otra causa, el sistema informa del error.

1.4. Sesión: Atravesar un campo de minas con una pieza

1.4.1 Curso normal de eventos:

Acción de los actores	Respuesta del sistema
1. El usuario eligió la opción de atravesar un campo de minas con una pieza	
	2. El sistema muestra un tablero vacío, y a la derecha del mismo todas las piezas de ajedrez, Piolín y 11 minas
3. El usuario puede elegir entre: a. Colocar una pieza en el tablero. b. Colocar una mina en el tablero c. Colocar a Piolín en el tablero d. Recuperar una configuración guardada.	

1.4.1.1. Sesión: Colocar una pieza en el tablero:

1.4.1.1.1. Curso normal de eventos:

	Acción de los actores		Respuesta del sistema
1.	El usuario eligió la opción de colocar una pieza en el tablero. Coloca una pieza.		
		2.	El sistema le da la opción de guardar la configuración inicial del tablero con un nombre.
3.	El usuario elige el nombre con el que quiere guardar dicha configuración		
		4.	El sistema genera un archivo de texto con la configuración inicial guardada. Hasta que el usuario no haya guardado las configuraciones de minas, piezas y Piolín, el sistema no le permitirá continuar.
5.	El usuario introduce el nombre del fichero con el que quiere guardar el ejercicio en formato HTML		
		6.	El sistema genera el ejercicio con las opciones deseada por el profesor

1.4.1.1.2. Cursos alternos:

Línea 4: Imposible guardar la configuración en la base de datos de configuraciones, se muestra un mensaje de error. Hasta que el usuario no haya guardado todos los datos necesarios para generar este ejercicio (configuración de piezas, minas y Piolín), el sistema no le permitirá guardar el ejercicio.

Línea 5: Si el usuario no introduce el nombre del fichero, el sistema da un mensaje que informa de ello.

Línea 6: Imposible guardar el ejercicio por falta de espacio u otra causa, el sistema informa del error.

1.4.1.2. Sesión: Colocar una mina en el tablero:

1.4.1.2.1. Curso normal de eventos:

Acción de los actores	Respuesta del sistema
1. El usuario eligió la opción de colocar una mina en el tablero. Coloca una mina.	
	2. El sistema le da la opción de guardar la configuración inicial del tablero con un nombre.
3. El usuario elige el nombre con el que quiere guardar dicha configuración	
	4. El sistema genera un archivo de texto con la configuración inicial guardada. Hasta que el usuario no haya guardado la configuración de las piezas, las minas y Piolín, no podrá continuar.
5. El usuario introduce todos los datos solicitados por el sistema	
	6. El usuario genera el ejercicio con las opciones deseada por el profesor

1.4.1.2.2. Cursos alternos:

Línea 4: Imposible guardar la configuración en la base de datos de configuraciones, se muestra un mensaje de error. Hasta que el usuario no haya guardado todos los datos necesarios para generar este ejercicio (configuración de piezas, minas y Piolín), el sistema no le permitirá guardar el ejercicio.

Línea 5: Si el usuario no introduce el nombre del fichero, el sistema da un mensaje que informa de ello.

1.4.1.3. Sesión: Colocar Piolín en el tablero:

1.4.1.3.1. Curso normal de eventos:

Acción de los actores	Respuesta del sistema
1. El usuario eligió la opción de colocar a Piolín en el tablero. Coloca a Piolín	

	2. El sistema le da la opción de guardar la configuración inicial del tablero con un nombre.
3. El usuario elige el nombre con el que quiere guardar dicha configuración	
	4. El sistema genera un archivo de texto con la configuración inicial guardada. Hasta que el usuario no haya colocado y guardado la configuración de las piezas, las minas y Piolín, no podrá continuar.
5. El usuario introduce todos los datos solicitados por el sistema	
	6. El usuario genera el ejercicio con las opciones deseada por el profesor

1.4.1.3.2. Cursos alternos:

Línea 4: Imposible guardar la configuración en la base de datos de configuraciones, se muestra un mensaje de error. Hasta que el usuario no haya guardado todos los datos necesarios para generar este ejercicio (configuración de piezas, minas y Piolín), el sistema no le permitirá guardar el ejercicio.

Línea 5: Si el usuario no introduce el nombre del fichero, el sistema da un mensaje que informa de ello.

1.4.1.4 Sesión: Recuperar una configuración guardada:

1.4.1.4.1. Curso normal de eventos:

Acción de los actores	Respuesta del sistema
1. El usuario eligió la opción de recuperar una configuración inicial guardada	
	2. El sistema solicita el nombre del archivo que contiene la configuración inicial guardada
3. El usuario escribe el nombre	

del fichero que quiere recuperar.	
	4. El sistema actualiza automáticamente el tablero con la configuración que ha recuperado de ese archivo de texto.
5. El usuario introduce todos los datos solicitados por el sistema	
	6. El usuario genera el ejercicio con las opciones deseada por el profesor

1.4.1.4.2. Cursos alternos:

Línea 3: Imposible recuperar la configuración de la base de datos de configuraciones, bien porque no existe o porque no deja acceder a esa copia, se muestra un mensaje de error.

Línea 6: Imposible generar el ejercicio, porque hay algún dato que el usuario no ha introducido. Se muestra un mensaje indicando que debe introducir todos los datos.

1.5. Sesión: Jugar una partida completa activando opciones

1.5.1 Curso normal de eventos:

Acción de los actores	Respuesta del sistema
1. El usuario eligió la opción de jugar una partida de ajedrez	
	2. El sistema muestra un tablero vacío y todas las piezas del ajedrez
3. El usuario puede elegir entre las siguientes acciones: a. Colocar una pieza en el tablero. b. Recuperar una configuración guardada previamente.	

1.5.1.1. Sesión: Colocar una pieza en el tablero:

1.5.1.1.1. Curso normal de eventos:

Acción de los actores	Respuesta del sistema
1. El usuario eligió la opción de colocar una pieza en el tablero. Coloca una pieza.	
	2. El sistema le da la opción de guardar la configuración inicial del tablero con un nombre.
3. El usuario elige el nombre con el que quiere guardar dicha configuración	
	4. El sistema genera un archivo de texto con la configuración inicial guardada. Muestra al usuario la configuración establecida y solicita la frase que quiere que aparezca. Elegir entre: a. Juego aleatorio. b. Juego valorando los porcentajes de ganar material y ganar el centro del tablero
5. El usuario introduce todos la frase y la opción deseada.	
	6. El sistema guarda los datos. Finalmente solicita al usuario un nombre para ejercicio generado.
7. El usuario introduce el nombre para ese ejercicio.	
	8. El sistema genera el ejercicio con las opciones deseada por el profesor. Tener en cuenta que el juego aleatorio no usa inteligencia, mientras que el otro si hace uso de inteligencia

1.5.1.1.2. Cursos alternos:

Línea 4: Imposible guardar la configuración en la base de datos de configuraciones, se muestra un mensaje de error.

Línea 6: Imposible guardar los datos. Si hay algún dato que el usuario no ha introducido, se muestra un mensaje indicando que debe introducir todos los datos.

Línea 7: Si el usuario no introduce el nombre del fichero, el sistema da un mensaje que informa de ello.

Línea 8: Imposible guardar el ejercicio por falta de espacio u otra causa, el sistema informa del error.

1.5.1.2 Sesión: Recuperar una configuración guardada:

1.3.1.2.1. Curso normal de eventos:

Acción de los actores	Respuesta del sistema
1. El usuario eligió la opción de recuperar una configuración inicial guardada	
	2. El sistema solicita el nombre del archivo que contiene la configuración inicial guardada
3. El usuario escribe el nombre del fichero que quiere recuperar.	
	4. El sistema actualiza automáticamente el tablero con la configuración que ha recuperado de ese archivo de texto. El sistema genera un archivo de texto con la configuración inicial guardada. Muestra al usuario la configuración establecida y solicita la frase que quiere que aparezca. Elegir entre: a. Juego aleatorio. b. Juego valorando los porcentajes de ganar material y ganar el centro del tablero
5. El usuario introduce todos la frase y la opción deseada.	
	6. El sistema guarda los datos. Finalmente solicita al usuario un nombre para ejercicio generado.

7. El usuario introduce el nombre para ese ejercicio.	
	8. El sistema genera el ejercicio con las opciones deseada por el profesor. Tener en cuenta que el juego aleatorio no usa inteligencia, mientras que el otro si hace uso de inteligencia

1.3.1.2.2. Cursos alternos:

Línea 3: Imposible recuperar la configuración de la base de datos de configuraciones, bien porque no existe o porque no deja acceder a esa copia, se muestra un mensaje de error.

Línea 6: Imposible guardar los tres ficheros si hay algún dato que el usuario no ha introducido, se muestra un mensaje indicando que debe introducir todos los datos.

Línea 7: Si el usuario no introduce el nombre del fichero, el sistema da un mensaje que informa de ello.

Línea 8: Imposible guardar el ejercicio por falta de espacio u otra causa, el sistema informa del error.

1.6. Sesión: Dar jaque en X jugadas

1.6.1 Curso normal de eventos:

Acción de los actores	Respuesta del sistema
1. El usuario eligió la opción de dar jaque en X jugadas	
	2. El sistema muestra un tablero vacío y todas las piezas del ajedrez
3. El usuario puede elegir entre las siguientes acciones: a. Colocar una pieza en el tablero. b. Recuperar una configuración guardada previamente.	

1.6.1.1. Sesión: Colocar una pieza en el tablero:

1.6.1.1.1. Curso normal de eventos:

Acción de los actores	Respuesta del sistema
1. El usuario eligió la opción de colocar una pieza en el tablero. Coloca una pieza.	
	2. El sistema le da la opción de guardar la configuración inicial del tablero con un nombre.
3. El usuario elige el nombre con el que quiere guardar dicha configuración	
	4. El sistema genera un archivo de texto con la configuración inicial guardada. Muestra al usuario la configuración establecida y solicita la pregunta a hacerle al alumno, y el número de movimientos en el que quiere dar jaque
5. El usuario introduce todos los datos solicitados por el sistema	
	6. El sistema guarda datos necesarios para el ejercicio
7. El usuario introduce el nombre para ese ejercicio.	
	8. El sistema genera el ejercicio con las opciones deseada por el profesor

1.6.1.1.2. Cursos alternos:

Línea 4: Si el usuario no ha colocado ambos reyes en el tablero, el sistema informará de ello.

Línea 7: Si el usuario no introduce el nombre del fichero, el sistema da un mensaje que informa de ello.

Línea 8: Imposible guardar el ejercicio por falta de espacio u otra causa, el sistema informa del error.

1.6.1.2 Sesión: Recuperar una configuración guardada:

1.6.1.2.1. Curso normal de eventos:

Acción de los actores	Respuesta del sistema
1. El usuario eligió la opción de recuperar una configuración inicial guardada	
	2. El sistema solicita el nombre del archivo que contiene la configuración inicial guardada
3. El usuario escribe el nombre del fichero que quiere recuperar.	
	4. El sistema actualiza automáticamente el tablero con la configuración que ha recuperado de ese archivo de texto.
5. El usuario introduce todos los datos solicitados por el sistema	
	6. El sistema guarda tres ficheros uno con la pregunta, y el número de movimientos en el que se quiere dar jaque
7. El usuario introduce el nombre para ese ejercicio.	
	8. El sistema genera el ejercicio con las opciones deseada por el profesor

1.3.1.2.2. Cursos alternos:

Línea 3: Imposible recuperar la configuración de la base de datos de configuraciones, bien porque no existe o porque no deja acceder a esa copia, se muestra un mensaje de error.

Línea 6: Imposible guardar los tres ficheros si hay algún dato que el usuario no ha introducido, se muestra un mensaje indicando que debe introducir todos los datos.

Línea 7: Si el usuario no introduce el nombre del fichero, el sistema da un mensaje que informa de ello.

Línea 8: Imposible guardar el ejercicio por falta de espacio u otra causa, el sistema informa del error.

2. Caso de uso: Ejercicio 1

2.1. Sesión: Principal

2.1.1. Caso de Uso: Ejercicio1. Marcar casillas a las que se puede mover una pieza.

2.1.2. Actores: Alumno

2.1.3. Propósito: Permitir al alumno aprender el movimiento de cada pieza

2.1.4. Resumen: Se muestra un tablero con una pieza y el alumno debe marcar todas las casillas a las que se puede mover esa pieza

2.1.5. Tipo: Primario y esencial

2.1.6. Referencias cruzadas:

2.1.6.1. Funciones: CrearCasilla, CrearPiezas, ColocarPiezas, MouseDown, MouseUp, MouseMove, Marcar, Desmarcar, PintarSolución

2.1.6.2. Casos de uso: Generar ejercicio

2.1.7. Curso normal de los eventos:

	Acción de los actores		Respuesta del sistema
1.	El usuario ha elegido jugar al ejercicio 1		
		2.	El sistema le proporciona un tablero con una pieza y le pregunta a qué casillas puede moverse dicha pieza
3.	El usuario ha de marcar sobre el tablero las casillas a las que dicha pieza puede moverse, y darle a comprobar solución		
		4.	El sistema muestra la solución correcta sobre el tablero dibujando las casillas solución con un círculo rojo y comprueba si la solución respondida por el usuario es correcta, si es así le da la enhorabuena. En caso contrario muestra un mensaje adecuado.

2.1.8. Cursos alternos:

Línea 4: Imposible mostrar la solución correcta por la imposibilidad de leer el archivo con la solución

3. Caso de uso: Ejercicio 2

3.1. Sesión: Principal

3.1.1. Caso de Uso: Ejercicio2. Elegir una respuesta entre varias posibles.

3.1.2. Actores: alumno

3.1.3. Propósito: Permitir al alumno adquirir conocimientos de ajedrez

3.1.4. Resumen: Se muestra un tablero con una configuración inicial, se hace una pregunta y se dan posibles opciones para la respuesta correcta, el alumno ha de marcar sólo una

3.1.5. Tipo: Primario y esencial

3.1.6. Referencias cruzadas:

3.1.6.1. Funciones: CrearCasilla, CrearPiezas, ColocarPiezas, MouseDown, MouseUp, MouseMove, MostrarSolución

3.1.6.2. Casos de uso: Generar ejercicio

3.1.7. Curso normal de los eventos:

Acción de los actores	Respuesta del sistema
1. El usuario ha elegido jugar al ejercicio 2	
	2. El sistema le proporciona un tablero con una configuración inicial, le formula una pregunta, le ofrece posibles respuestas dicha pieza
3. El usuario ha de marcar una de las posibles respuestas, y darle al botón de comprobar solución para ver si lo está haciendo bien	
	4. El sistema muestra la opción correcta.

3.1.8. Cursos alternos:

Línea 2: Imposible mostrar la pregunta y las posibles respuestas
Línea 4: Imposible mostrar la solución correcta por la imposibilidad de leer el archivo con la solución

4. Caso de uso: Ejercicio 3

4.1. Sesión: Principal

4.1.1. Caso de Uso: Ejercicio3. Mover una pieza por un campo minado.

4.1.2. Actores: Alumno

4.1.3. Propósito: Permitir al alumno perspicacia para mover las distintas piezas por el tablero sorteando posibles obstáculos

4.1.4. Resumen: Se muestra un tablero inicial con una pieza, Piolín y varias minas, el alumno debe de llegar con la pieza hasta donde esté Piolín sorteando todas las minas.

4.1.5. Tipo: Primario y esencial

4.1.6. Referencias cruzadas:

4.1.6.1. Funciones: CrearCasilla, CrearPiezas, ColocarPiezas, MouseDown, MouseUp, MouseMove, PintarMina, PintarMeta

4.1.6.2. Casos de uso: Generar ejercicio

4.1.7. Curso normal de los eventos:

Acción de los actores	Respuesta del sistema
1. El usuario ha elegido jugar al ejercicio 3	
	2. El sistema le proporciona un tablero con minas esparcidas por él, una pieza y Piolín (meta)
3. El usuario ha de ir avanzando con la pieza hasta llegar a Piolín sin pasar por encima de una mina	
	4. El sistema comprueba si los movimientos que hace el jugador con la pieza correspondiente son válidos, va mostrando el número de movimientos en todo momento
5. El usuario llega hasta la casilla donde está Piolín	
	6. El sistema indica el número de movimientos que el jugador ha tardado en

alcanzar la meta

4.1.8. Cursos alternos:

Línea 4: Si el usuario no realiza un movimiento válido con la pieza correspondiente el sistema no actualiza ese movimiento.

5. Caso de uso: Ejercicio 4

5.1. Sesión: Principal

5.1.1. Caso de Uso: Ejercicio4. Jugar una partida completa activando opciones.

5.1.2. Actores: Alumno

5.1.3. Propósito: Permitir a un alumno demostrar todo lo aprendido practicándolo en una partida.

5.1.4. Resumen: Se juega una partida el alumno contra la máquina.

5.1.5. Tipo: Primario y esencial

5.1.6. Referencias cruzadas:

5.1.6.1. Funciones: CrearCasilla, CrearPiezas, ColocarPiezas, MouseDown, MouseUp, MouseMove, EstimarJugada

5.1.6.2. Casos de uso: Generar ejercicio

5.1.7. Curso normal de los eventos:

Acción de los actores	Respuesta del sistema
1. El usuario ha elegido jugar al ejercicio 4, se muestran varias opciones: a. Jugar una partida "inteligente" b. Jugar aleatoriamente c. Jugar a ganar material d. Jugar a conquistar el centro del tablero	
	2. El sistema trata correctamente la opción elegida

5.2. Sesión: Jugar una partida "inteligente"

5.2.1. Curso normal de eventos:

Acción de los actores	Respuesta del sistema
1. El usuario eligió la opción de jugar una partida inteligente	
	2. El sistema muestra una opción con todas las piezas
3. El usuario comienza a mover	
	4. El sistema analiza la jugada, estima los movimientos y le responde con el mejor movimiento posible.

5.2.2. Cursos alternos

Línea 3 y 4: Los pasos 3 y 4 se repiten hasta que el usuario o el sistema ganen la partida o queden tablas.

5.3. Sesión: Jugar aleatoriamente

5.3.1. Curso normal de eventos:

Acción de los actores	Respuesta del sistema
1. El usuario eligió la opción de jugar aleatoriamente	
	2. El sistema muestra un tablero con las piezas del ajedrez colocadas sobre él
3. El usuario comienza a mover	
	4. El sistema elige de forma aleatoria uno de los posibles movimientos.

5.3.2. Cursos alternos

Línea 3 y 4: Los pasos 3 y 4 se repiten hasta que el usuario o el sistema ganen la partida o queden tablas.

5.4. Sesión: Jugar a ganar material

5.4.1. Curso normal de eventos:

Acción de los actores	Respuesta del sistema
1. El usuario eligió la opción de jugar a ganar material	
	2. El sistema muestra un tablero con todas las piezas colocadas sobre él
3. El usuario comienza a mover la pieza deseada	
	4. El sistema elige el movimiento más adecuado "inteligentemente", teniendo en cuenta que juega a ganar material, con lo cual valorará más las jugadas con las que más material gane o las jugadas que más material le coman al contrario.

5.4.2. Cursos alternos

Línea 3 y 4: Los pasos 3 y 4 se repiten hasta que el usuario o el sistema ganen la partida o queden tablas.

5.5. Sesión: Jugar a conquistar el centro del tablero

5.5.1. Curso normal de eventos:

Acción de los actores	Respuesta del sistema
1. El usuario eligió la opción de jugar a conquistar el centro del tablero	
	2. El sistema muestra un tablero con las piezas de ajedrez colocadas sobre él
3. El usuario comienza a mover la pieza deseada	
	4. El sistema responde con el mejor movimiento teniendo en cuenta que el objetivo es conquistar el centro del tablero con lo cuál valorará

	más los movimientos que conquisten el centro del tablero.

5.2.2. Cursos alternos

Línea 3 y 4: Los pasos 3 y 4 se repiten hasta que el usuario o el sistema ganen la partida o queden tablas.

6. Caso de uso: Ejercicio 5

6.1. Sesión: Principal

6.1.1. Caso de Uso: Ejercicio5. Dar jaque en X jugadas

6.1.2. Actores: Alumno

6.1.3. Propósito: Permitir a un alumno aprender a atacar al rey enemigo.

6.1.4. Resumen: Se muestra un tablero con una configuración inicial y el jugador blanco ha de dar jaque al jugador negro en el menor número de jugadas posible.

6.1.5. Tipo: Primario y esencial

6.1.6. Referencias cruzadas:

6.1.6.1. Funciones: CrearCasilla, CrearPiezas, ColocarPiezas, MouseDown, MouseUp, MouseMove, InicializarContador, IncrementarContador

6.1.6.2. Casos de uso: Generar ejercicio.

6.1.7. Curso normal de los eventos:

Acción de los actores	Respuesta del sistema
1. El usuario ha elegido jugar al ejercicio 5	
	2. El sistema le proporciona un tablero con una configuración inicial determinada
3. El usuario tiene que dar jaque al rey contrario en X movimientos	
	4. El sistema comprueba si los movimientos que hace el jugador con la pieza

	correspondiente son válidos, va mostrando el número de movimientos en todo momento
5. El usuario deja de mover cuando el número de movimientos es X	
	6. El sistema indica si la prueba se ha superado con éxito o hay que seguir practicando

6.1.8. Cursos alternos:

Línea 4: Si el usuario no realiza un movimiento válido con la pieza correspondiente el sistema no actualiza ese movimiento.

5. Decisiones Tecnológicas

5.1. Planteamiento inicial

Como se viene diciendo, para que el sistema cumpla con los requisitos especificados se necesitan dos componentes bien diferenciados: por un lado la herramienta generadora de páginas HTML como contenedoras de Applets, y por otro lado es necesario crear los propios Applets para que puedan ser utilizados desde estas páginas.

Se pueden adoptar varias resoluciones al respecto de las tecnologías a emplear para cada una de estas componentes así como sobre la forma de generarlas.

En una primera valoración, se barajó la posibilidad de que fuese la herramienta generadora de las páginas la que crease el propio código de las clases necesarias para el funcionamiento de los Applets. Esto conlleva la necesidad por parte del usuario de un compilador java para poder crear los archivos '.class' que necesita el Applet para funcionar. Esta medida podría restringir el espectro de usuarios a los que va dirigida la herramienta ya que implica conocimientos básicos de programación, además de hacerla menos apetecible para los posibles clientes al obligarles a hacer trabajo "extra" con tecnologías que pueden no tener a su disposición. Se optó por consiguiente por trabajar desde dos planos de acción: la herramienta generadora de Applets sólo crearía la codificación HTML que va dentro de la página que alberga al Applet mientras que el código de los Applets propiamente dicho sería estático, es decir, se proporcionaría al usuario en código máquina parametrizándolo lo suficiente como para dotar al resultado de la máxima flexibilidad posible.

Nunca se tuvo en cuenta otra posibilidad que no fuera la del formato Applet para el resultado final del sistema. Esto obliga necesariamente al empleo de tecnologías Java para su construcción. La herramienta generadora de páginas HTML no tenía ninguna restricción en principio, lo que permitió tener una mayor libertad a la hora de elegir el modo de implementarla. Después de valorar los pros y los contras – que se expondrán en el siguiente apartado – se optó por el uso del lenguaje de programación C++ para llevarla a cabo.

5.2. Estudio comparativo de las tecnologías posibles

Movidos por la ambición de hacer que la herramienta fuera lo más multiplataforma, rápida y eficiente posible, se propuso hacer un estudio comparativo entre las dos tecnologías más potentes y utilizadas actualmente en el mercado: Java y C++. El resultado se muestra en la tabla que viene a continuación:

	A favor	En contra
Java	• Reutilización: dado que, como se dijo dicho anteriormente, la exigencia de que el resultado final se presentase en formato Applet obliga a usar Java para la generación de éstos, emplear también tecnología Java en el desarrollo de la herramienta generadora de páginas facilitaría la reutilización de las clases más elementales tales como las que representan a los cuadrados de un tablero, las piezas de éste o incluso el tablero en su composición más genérica. Además de estas cuestiones propias del diseño, muchas otras más algorítmicas – movimientos de las piezas, comprobación de situación de jaque, enroque ...- podrían utilizarse en caso de ser necesario en la herramienta. Esta reutilización no solo afectaría al código generado, sino también a cualquier otra posible fuente externa que se pudiera encontrar – por ejemplo a través de la red – y que se pudiese servir para nuestros propósitos. • Portabilidad: uno de los grandes logros de Java es justamente su capacidad de adaptarse a distintas plataformas hardware. Esto se ajustaba a nuestros objetivos de hacer la herramienta lo más multiplataforma posible.	• Ineficiencia: en general, las aplicaciones desarrolladas en Java consumen bastantes recursos, lo que se traduce en una ralentización del programa en tiempo de ejecución. • Interfaces de usuario costosas: aunque gracias a la aparición de nuevas herramientas como el JBuilder o el JCreator, que incluyen ayudas para la creación de interfaces gráficas, la construcción de elementos de interacción gráficos con el usuario ha mejorado sensiblemente en cuanto a simplicidad y eficiencia, sigue siendo un trabajo bastante tedioso para los programadores crear una G.U.I en Java. La propia filosofía que subyace al lenguaje Java y que es muy ventajosa en algunos aspectos (generalidad, modularidad, portabilidad...) hace que sea necesario escribir mucho código para gestionar los elementos gráficos y sus eventos.
C++	• Eficiencia: en contraposición a las herramientas desarrolladas en Java, las escritas en C++ tienen un menor consumo de recursos, lo que las hace más eficientes en este sentido y por consiguiente, en el de tiempo de ejecución. Interfaces de usuario amigables: gracias a la iniciativa de herramientas como el C++ Builder que combinan la potencia del lenguaje C++ con las interfaces gráficas de Delphi, crear G.U.I con C++ se convierte en una tarea sencilla para el programador. Además el aspecto de las aplicaciones desarrolladas bajo estos entornos es muy similar a cualquier aplicación típica de Windows, lo que contribuye a que el usuario se pueda sentir más familiarizado con ellas en una primera toma de contacto.	• Repetición de código: usar tecnologías distintas para la herramienta y para la creación de Applets obligaría a generar código idéntico en dos lenguajes distintos. En principio, el salto de Java a C++ no es muy grande, por lo que podría no constituir un serio problema, salvo cuando se trata de traducir algoritmos complejos que utilizan estructuras de datos exclusivas de la tecnología Java.

5.3. Decisiones adoptadas

A la hora de elegir entre una de las dos opciones había que sopesar qué compensaba más, si el esfuerzo de tener que escribir el mismo código en los dos lenguajes o bien la eficiencia y rapidez de la aplicación final. Ayudó a tomar esta decisión una componente que se encontró en Internet para el compilador C++ Builder: TChessBoard. Instalándola se podría utilizar en el proyecto y trabajar con ella como si fuese un elemento más de la paleta de componentes – como un botón o una etiqueta, por ejemplo – con la ventaja adicional de que todas las funcionalidades que se necesitaba que tuviese el tablero de ajedrez que debería ir en la herramienta estaban ya implementadas en la componente en forma de propiedades o bien formando parte del conjunto de métodos de la misma. No sería necesario pues repetir código en Java y en C++. Bastaría con implementarlo en Java para el funcionamiento de las Applets y aprender a manejar la componente encontrada.

Resuelto este aspecto, parecía bastante obvio emplear C++ como lenguaje de programación para implementar la herramienta, primando así la comodidad y eficacia del sistema para el usuario final sobre el resto de los aspectos. Este es por tanto el camino que se decidió seguir.

5.4. Conclusiones

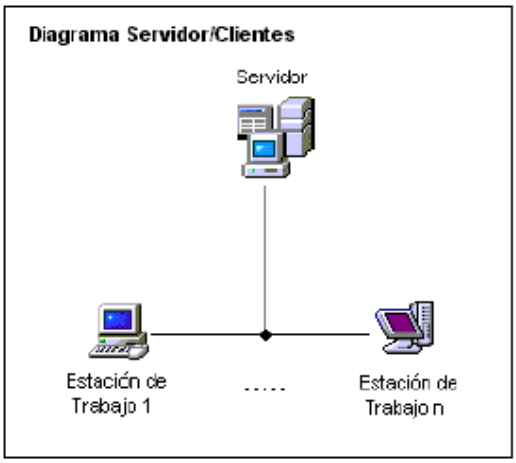
Los resultados obtenidos han sido bastante satisfactorios después de haber tomado la elección de usar C++ para implementar la herramienta generadora de páginas HTML y ha sido fácil y cómodo llevarla a cabo, gracias en gran parte, al uso de la componente TChessBoard.

Por otra parte, la interacción con el usuario se realiza de una manera sencilla en un entorno amigable, lo que hace del sistema una herramienta apetecible de usar para cualquier posible cliente siempre dentro de los objetivos para los que fue diseñada.

Es en el uso de Java para crear las Applets en lo que se ha discrepado un poco. Es evidente que es necesario el uso de este lenguaje para crear los elementos Applet pero se considera que hubiera sido más eficiente haber creado alguna estructura o herramienta subyacente escrita en algún lenguaje más eficiente – como C++ - para realizar el trabajo 'duro' del Applet, por decirlo así. La recogida automática de basura en Java actúa en nuestra contra cuando se generan grandes estructuras de datos que se

quieren destruir en cuanto son innecesarias para que no consuman recursos y disminuyan la rapidez en ejecución. Como se verá más adelante, los resultados obtenidos para los Applets no han sido todo lo buenos que se hubiese esperado en este sentido y se cree que el uso combinado de estas dos tecnologías para tal fin hubiera podido paliar tales deficiencias.

Como posible ampliación al proyecto sería conveniente que permitiese una comunicación entre distintos PCs. Uno de ellos sería el servidor (que sería el profesor), encargado de generar los ejercicios y los otros serían los clientes (alumnos), que desarrollarían sus ejercicios y se los enviarían inmediatamente al profesor para poder llevar un control absoluto de todos sus alumnos. Se muestra a continuación un esquema deseado de lo que hubiera gustado que hubiese sido la aplicación, aunque por falta de tiempo, esta parte no se haya podido llevar a cabo.



6. Ciclo de vida

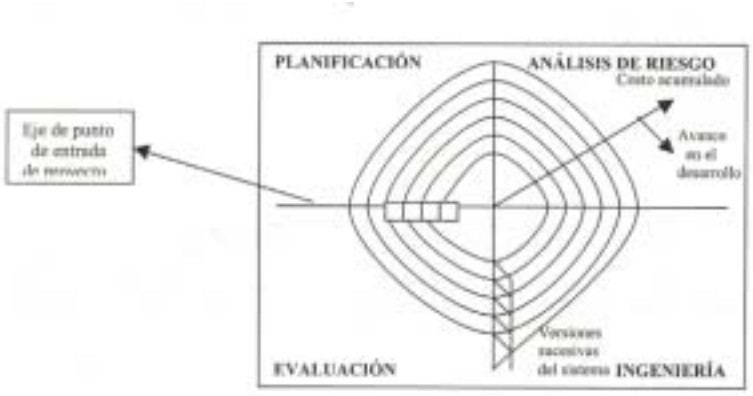
6.1 Elección del modelo de ciclo de vida

Se plantea la necesidad de establecer un ciclo de vida a seguir en el desarrollo del software del sistema. Para ello se valoran dos aspectos que se consideran fundamentales en la elección:

- la limitación en el tiempo
- la carencia de una especificación concreta y detallada sobre la forma que debía tener el producto final, lo que permitiría ir añadiendo o quitando detalles sobre el mismo según los resultados conseguidos y el ritmo de trabajo.

Esto llevó a pensar en un *modelo evolutivo* como el más adecuado para el proyecto, ya que se define conforme a la evolución del software con el tiempo, de forma que el desarrollo se realiza basándose en versiones que cada vez son más completas hasta llegar al software final.

Dentro de los modelos evolutivos disponibles se optó por el *modelo en espiral:*



En el *modelo en espiral* el software se desarrolla en una serie de versiones incrementales, de forma que en las primeras iteraciones, la versión incremental podría ser un prototipo y en las últimas iteraciones, se producen versiones cada vez más completas del software. De esta forma se proporciona el potencial necesario para el desarrollo rápido de versiones incrementales del software.

Consta de una serie de ciclos, cada uno de los cuales comienza identificando los objetivos, las alternativas y las restricciones del ciclo. Una vez evaluadas las alternativas respecto a los objetivos y teniendo en cuenta las restricciones, se lleva a cabo el ciclo correspondiente para una vez finalizado empezar a plantear el próximo.

El ciclo de iteración del modelo evolutivo se convierte en una espiral al añadir como dimensión radial una indicación del esfuerzo total realizado hasta cada momento, que será un valor siempre creciente.

El modelo en espiral es un enfoque realista del desarrollo de sistemas y de software a gran escala. Como el software evoluciona, a medida que progresa el proceso se comprende y reacciona mejor ante riesgos en cada uno de los niveles evolutivos. Utiliza la construcción de prototipos como mecanismo de reducción de riesgos, permitiendo a quien lo desarrolla aplicar el enfoque de construcción de prototipos en cualquier etapa de evolución del producto.

Por todas estas razones, se pensó que es el que mejor se ajusta a las necesidades del proyecto, y por esto se eligió como ciclo de vida a seguir, descartando a otros de la misma familia porque es el único que ofrece libertad absoluta respecto al número de iteraciones y por su capacidad de ir corrigiendo y ampliando los requisitos de las iteraciones anteriores.

6.2 Ciclos de vida del proyecto

A lo largo de la ejecución del proyecto, el sistema ha pasado por distintos ciclos de desarrollo, algunos de ellos se han solapado, o bien se han realizado en paralelo por los miembros integrantes del equipo. A grandes rasgos, estos ciclos se pueden resumir en:

Ciclo 1: Noviembre 2002 – Diciembre 2002

En este primer ciclo del proyecto, el objetivo fundamental a cubrir era el de extraer una especificación concreta y detallada del sistema a realizar a partir del planteamiento inicial del problema, analizando e identificando qué busca el cliente, cómo se puede llevar a cabo, qué requisitos debe cumplir el producto final y los medios que se debe poner para que se cumplan –elección de tecnologías a emplear-.

Para extraer esta información, se llevó a cabo una tarea de análisis y especificación de requisitos a través de repetidas entrevistas con el tutor de proyecto.

Una vez identificados los componentes del sistema (la herramienta generadora de ejercicios por un lado, y por otro los cinco tipos de ejercicios distintos que debía ser capaz de generar ésta), se pasó a una fase paralela de recogida de información. A través de distintas fuentes como Internet o libros didácticos de ajedrez para niños, se buscaron todos aquellos elementos que podrían ser de utilidad, tanto en la implementación directa de cada una de las partes del sistema (componentes de ajedrez, librerías, aplicaciones completas para jugar una partida de ajedrez en Java y C++…) como en la forma que deberían tener los ejercicios generados en cuanto a presentación, indicaciones para resolverlos, información necesaria para cada ejercicio…, tomando como referencia programas de carácter educativo en el campo del ajedrez como el 'ChessMaster' o visitando los sitios Web de organizaciones especializadas en esta área de enseñanza.

Como resultado de este ciclo se eligió como tecnología para la herramienta generadora, C++ y para los ejercicios generados, Java. La búsqueda de información condujo hasta la componente TChessBoard y a otros programas que se usaron en fases posteriores de desarrollo – SimpleChess, AppletJava... -.

Ciclo 2: Diciembre 2002 – Enero 2003

Con las ideas ya más claras sobre los aspectos que debía cubrir el proyecto y de los medios disponibles para hacerlo, se pasó a una primera fase de diseño e implementación en la que se optó por hacer un desarrollo paralelo de la herramienta generadora de ejercicios y de un ejercicio de prueba.

El primer prototipo de la herramienta generadora de ejercicios consistía en una pequeña aplicación capaz de dibujar un tablero y un conjunto de piezas a colocar en el tablero. Las funcionalidades eran muy básicas y se componían principalmente de: guardar la configuración de las piezas dentro del tablero, recuperar una configuración del tablero guardada previamente y crear un tablero nuevo que no contuviese ninguna pieza en su interior.

Para el prototipo de ejercicio se optó por algo sencillo y que fuera a servir para el resto de los ejercicios: un Applet capaz de dibujar un tablero y una serie de piezas dentro, que permitiese el movimiento de éstas en

cualquier dirección sin tener en cuenta las reglas que deben observar en una partida normal de ajedrez.

Los resultados obtenidos no fueron muy buenos en ninguno de los dos casos, pero sirvió a modo de acercamiento a la tecnología Applet – desconocida para el equipo de desarrollo hasta el momento – y de reencuentro con la tecnología C++.

Destacar de este ciclo la dificultad en la tarea de visualizar los tableros y las piezas correctamente en los navegadores, lo que consumió gran parte del tiempo de la iteración.

Ciclo 3: Febrero 2003 – Marzo 2003

A pesar de la aparente sencillez del objetivo marcado en el ciclo anterior, resultó algo complicado llevarlo a cabo, principalmente lo referido al prototipo del ejercicio. El desconocimiento en cuanto a la forma de funcionar de los Applets, hizo avanzar menos de lo deseado en este aspecto. Por consiguiente, se tomó la determinación de dedicarse exclusivamente en este ciclo a la creación de ejercicios, dejando a la herramienta generadora a un lado por el momento.

En este ciclo se trabajó en dos frentes paralelos. Por un lado se abordó la construcción de ejercicios del tipo 1 y 2 es decir, los que no requerían un movimiento correcto de las piezas dentro del tablero – marcar casillas, marcar una respuesta – y por otro la de los ejercicios 3, 4 y 5 que exigen que las piezas se muevan adecuadamente por las casillas del tablero.

La mayor dificultad que se encontró en este ciclo, fue sin duda, la de generar el código que controla que las piezas realicen movimientos correctos, lo que llevó algo más de tiempo de lo que se pensó. Había que valorar situaciones que al principio no se habían tenido en cuenta como el enroque, la promoción de los peones, comer al paso…

Al término de este ciclo ya se tenía los primeros prototipos de los tres primeros tipos de ejercicios, y un Applet que permitía jugar al ajedrez uno contra uno y que sería la plataforma de partida para los otros dos tipos que quedaban pendientes.

El resultado obtenido se aproximaba bastante a la idea inicial aunque surgieron los primeros problemas de compatibilidad con los distintos navegadores produciéndose retrasos en el desarrollo. Algunos

aspectos quedaron sin perfilar del todo, dejándolo para posteriores mejoras si el tiempo lo permitía – tal como la forma de arrastre de las piezas -.

Ciclo 4: Abril 2003 – Mayo 2003

La meta a alcanzar que se planteó al inicio de este ciclo de desarrollo fue la de que el computador lograse responder a los movimientos que un aprendiz hiciese sobre un tablero de ajedrez de manera inteligente.

Como primer paso para conseguir este fin, se hizo que el computador respondiese de manera a aleatoria los movimientos del contrincante, calculando los movimientos posibles a realizar y ejecutando uno de ellos.

Avanzando un poco más, se pretendió dotar al ejercicio de una cierta 'inteligencia' empleando una de las técnicas más usadas en teoría de juegos: un árbol Minimax con poda alfa-beta. Ello requería de una función de estimación que valorase las distintas jugadas para que el computador eligiese la más ventajosa para él. Fue ésta una tarea complicada que consumió la mayor parte del tiempo de este ciclo y que supuso la búsqueda de información, en la Web, principalmente, sobre cómo valorar en un tablero sus distintos elementos para modelizar mediante un número la bondad de la configuración para que el computador pudiese ganar la partida.

A la vez que se cubría este aspecto del sistema, se revisaron, corrigieron y mejoraron otros aspectos que ya fueron tratados en ciclos anteriores sobre los ejercicios 1, 2 y 3.

El resultado en esta iteración no acaba de responder a las especificaciones solicitadas. La respuesta del computador en el transcurso de la partida, no sigue las pautas que se predefinieron en las fases preliminares: las jugadas escogidas por el algoritmo Minimax no son las que se ajustan a las estrategias solicitadas por el usuario. De igual manera, los tiempos de respuesta exceden de las expectativas iniciales.

Ciclo 5: Junio 2003

Con los prototipos de los ejercicios ya finalizados, se volvió a ala implementación de la herramienta generadora que se había dejado aparcada en el segundo ciclo. La ausencia de una especificación precisa del cliente sobre la forma que debiera tener ésta, permitió tener una mayor libertad en su implementación.

Paralelamente a la construcción de la herramienta se empezó ya a generar la documentación del trabajo realizado en la elaboración del proyecto y siempre corrigiendo los errores pendientes del trabajo de otros ciclos anteriores.

Al término del mismo, se disponía ya de una versión de la herramienta bastante avanzada y un ligero esbozo de lo que debería ser la estructura general del documento presente.

Ciclo 6: Junio 2003 – Julio 2003

En este último ciclo de vida de construcción del software el esfuerzo se concentró en terminar la documentación del proyecto, por un lado, y en modificar el comportamiento y/o implementación de la herramienta generadora de ejercicios junto con el de los propios ejercicios para adaptarlos a los resultados que el cliente pensaba obtener.

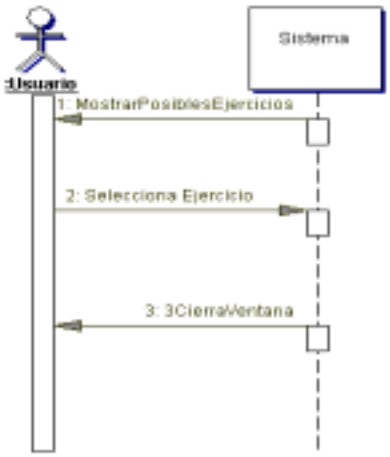
Fue un ciclo éste en el que el software debía estar ya 'a punto' para su entrega al cliente y en él se realizó la evaluación de la aplicación y corrección de errores.

Por último, con el objeto de que los posibles clientes tuvieran algún documento de referencia al que poder recurrir para poder apreciar los resultados de la herramienta antes de probarla ellos mismos, nos propusimos crear un página Web que, a modo de sencillo manual didáctico de ajedrez para niños, nos permitiera integrar varios ejercicios creados por la propia herramienta en un entorno adecuado para ellos. De esta manera los clientes tendrán, no sólo una visión completa de lo que pueden crear con nuestra herramienta, sino que también se harán una idea de la manera más adecuada de hacer uso de los ejercicios generados por ella.

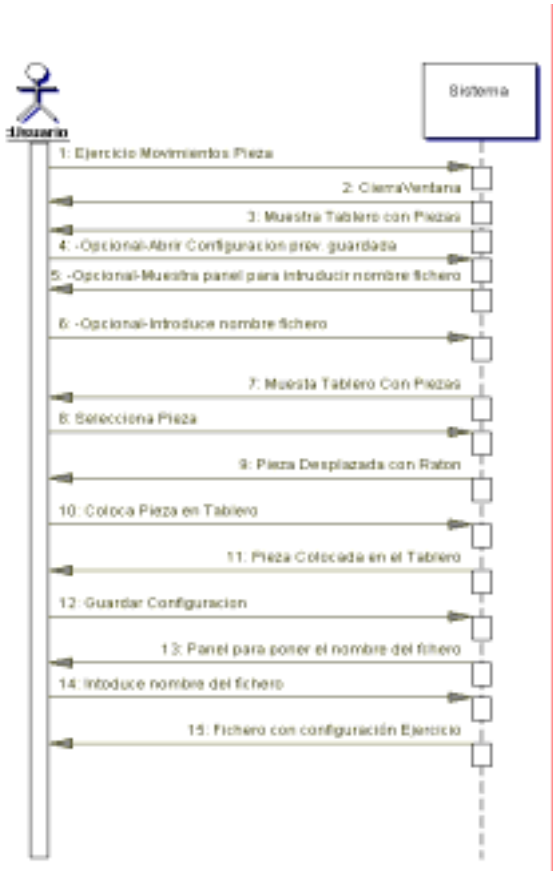
7. Análisis

El análisis se ha realizado teniendo en cuenta los casos de uso identificados. Para cada caso de uso mostramos el diagrama de secuencia correspondiente. Se muestran a continuación ordenados por los casos de uso.

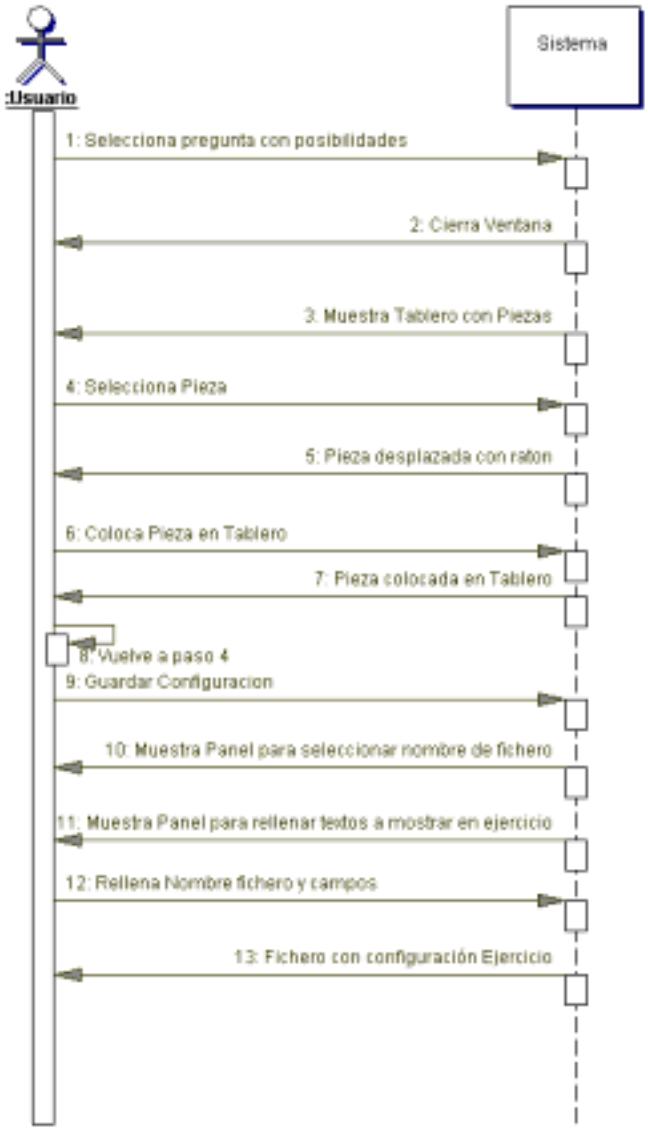
Caso de Uso 1: Generar ejercicio



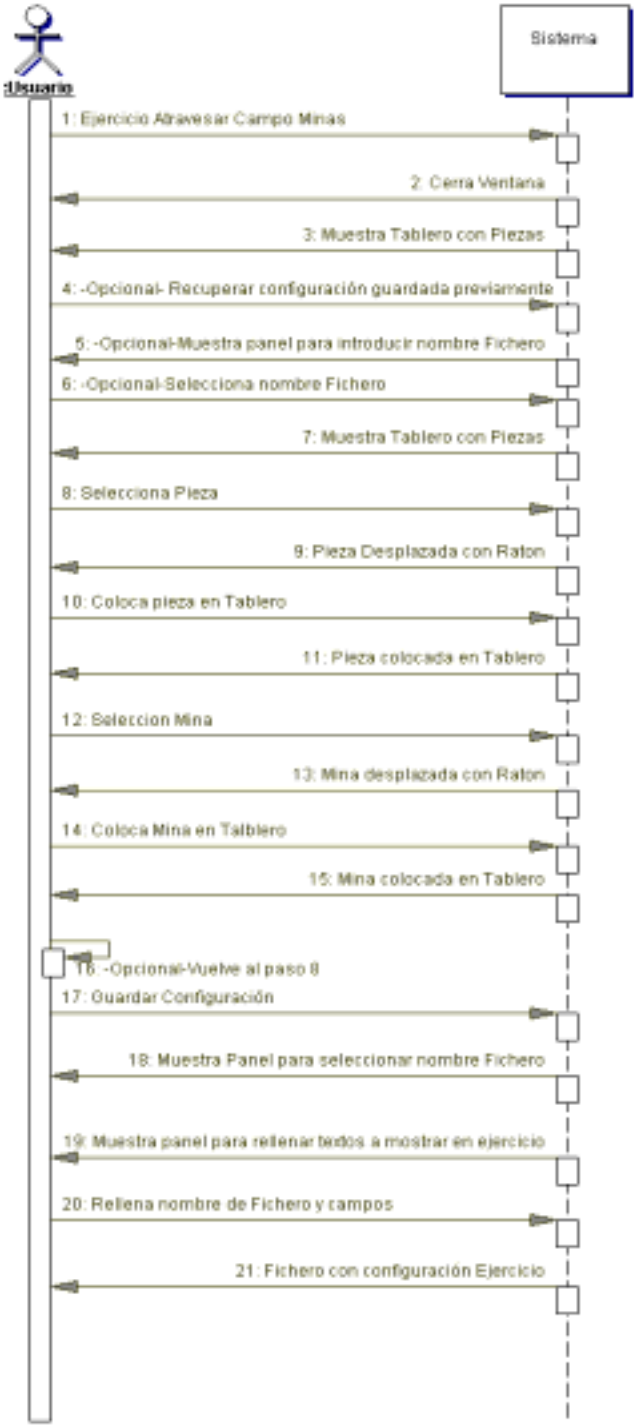
Caso de uso 1.2: Marcar los posibles movimientos de una pieza



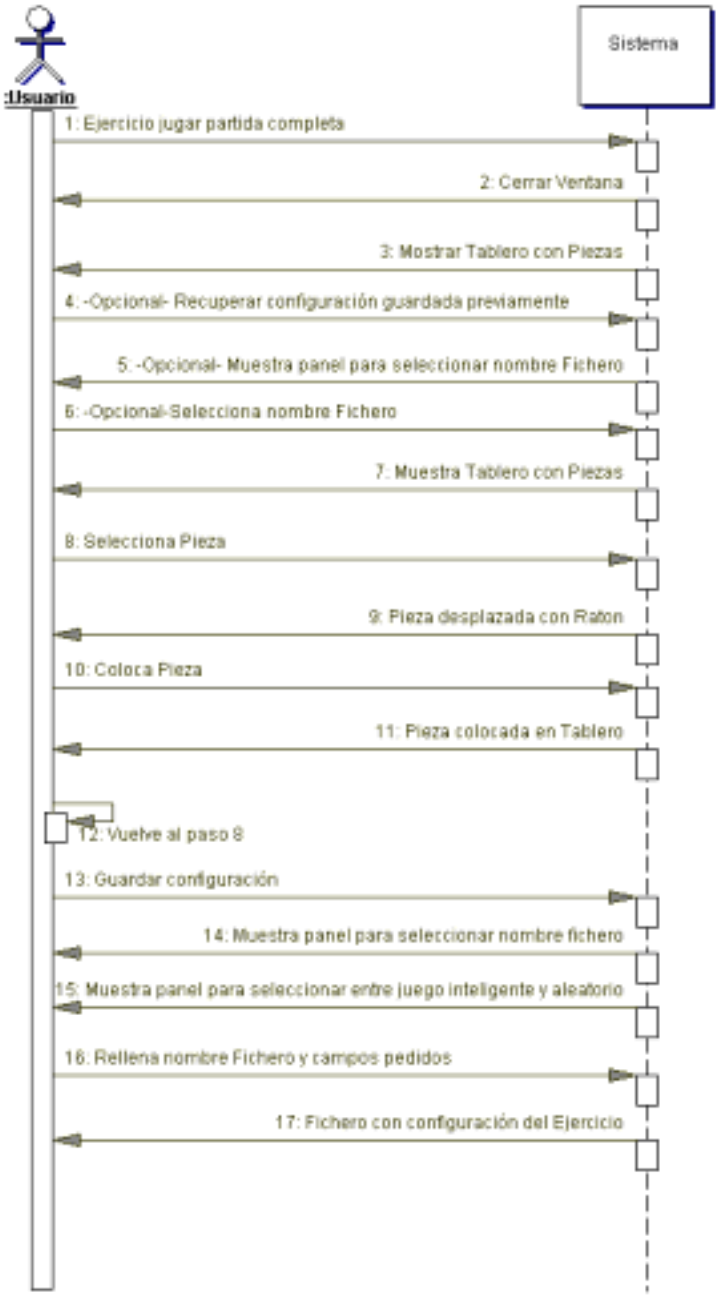
Caso de uso 1.3: Hacer una pregunta a partir de una configuración de tablero ofreciendo varias posibilidades



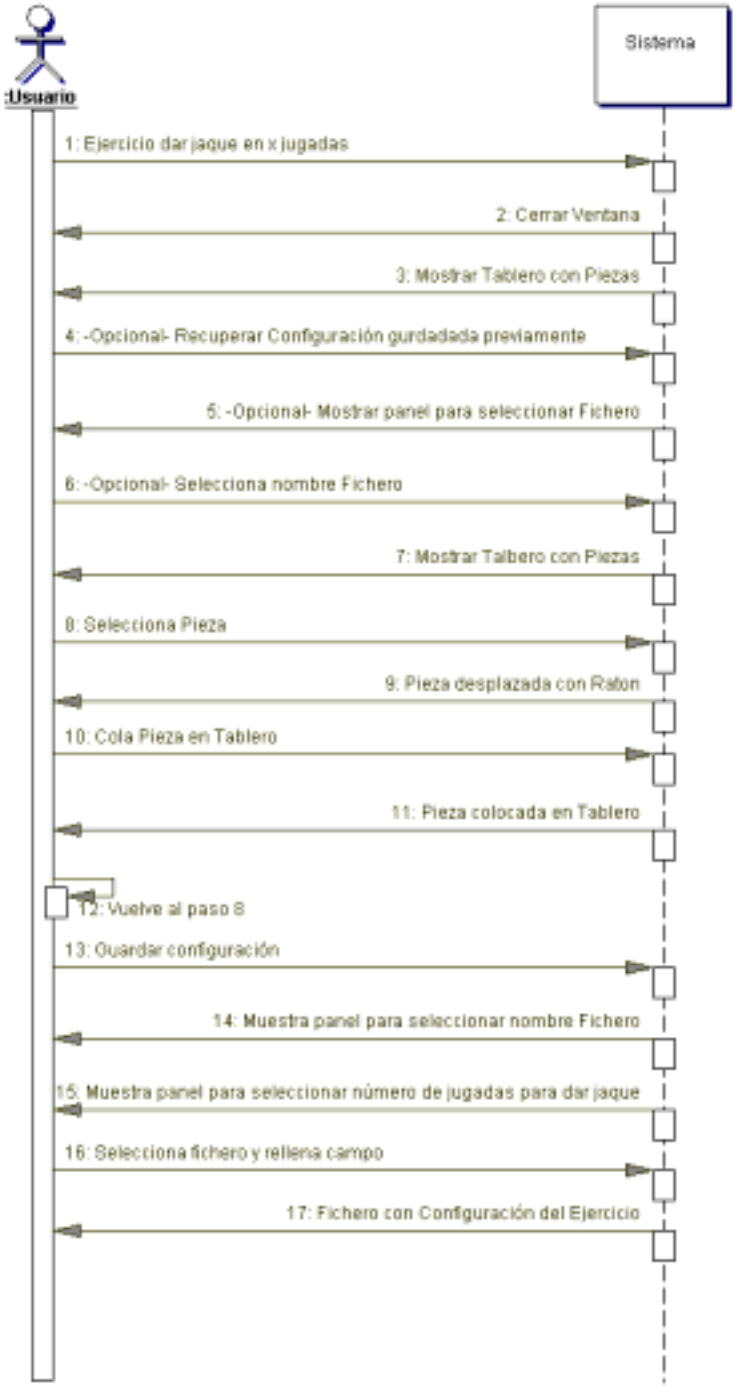
Caso de uso 1.4: Atravesar un campo de minas con una pieza



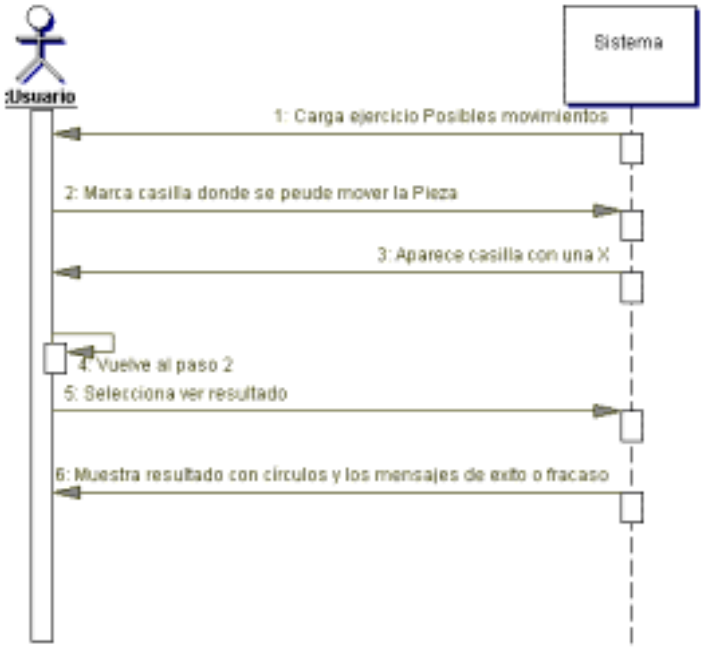
Caso de uso 1.5: Jugar una partida completa activando opciones



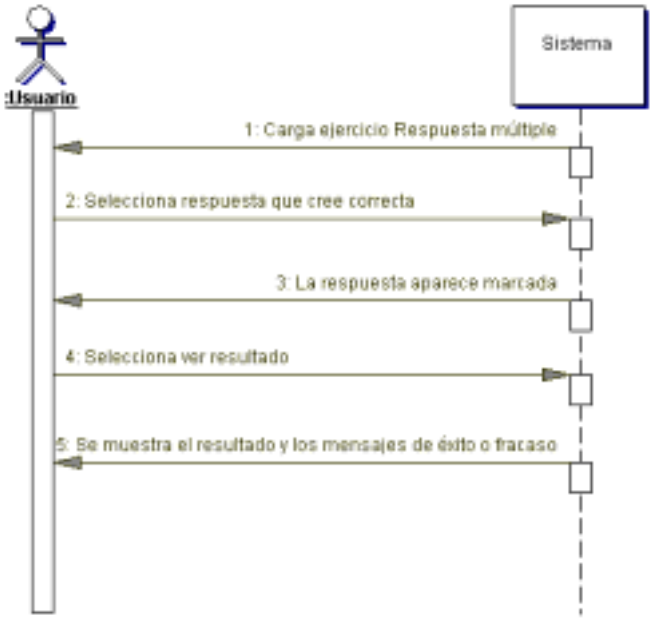
Caso de uso 1.6: Jaque en X jugadas



Caso de uso 2: Ejercicio1

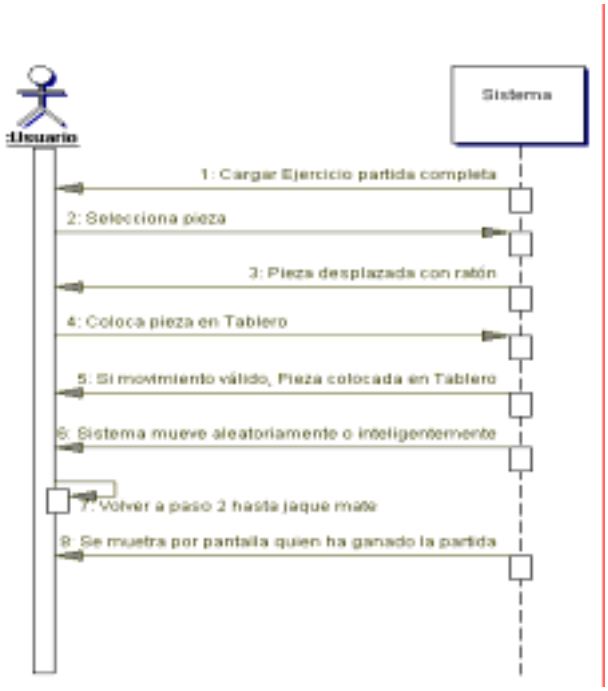


Caso de uso 3: Ejercicio2

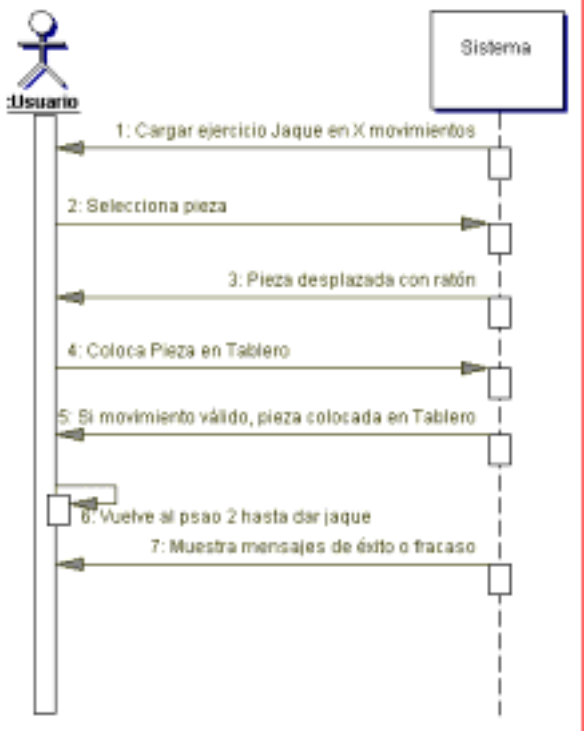


Caso de uso 4: Ejercicio3

Caso de uso 5: Ejercicio4



Caso de uso 6: Ejercicio5

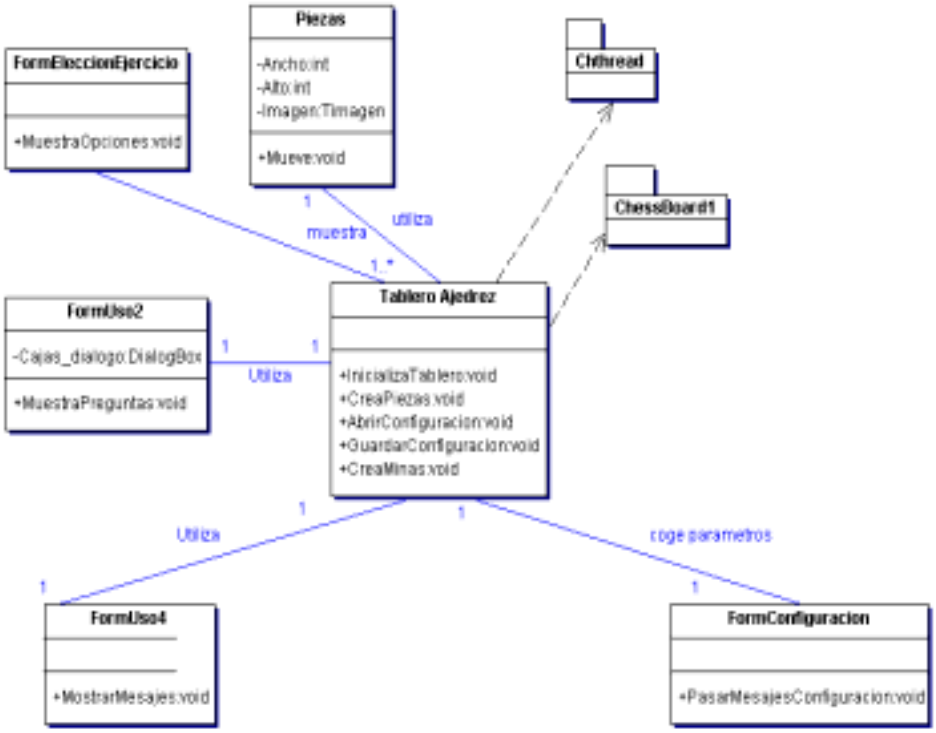


8. Diseño

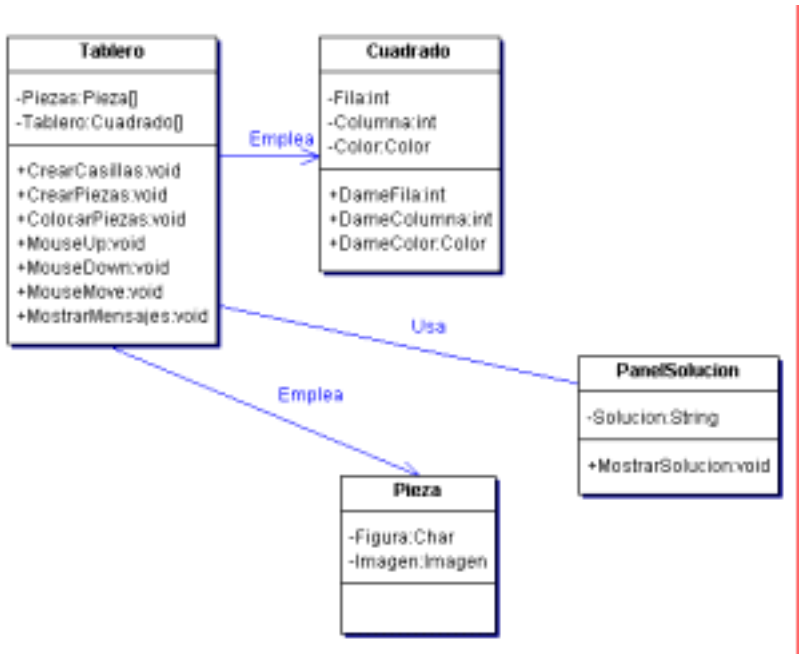
Para el diseño, al igual que en el análisis, se ha partido de los casos de uso para realizar los diagramas de clases. A partir de las funcionalidades que se han necesitado, se han ido añadiendo métodos a las clases conceptuales. El tipo de diagrama de clases que se ha decidido utilizar es UML por estar muy extendido y ser relativamente cercano a la estructura de clases de un lenguaje orientado objetos. Hay que destacar que el diagrama que se obtiene no deja de ser un diagrama conceptual, por lo que a la hora de implementar suelen aparecer otras clases y operaciones.

A continuación se muestra el diagrama de clases obtenido para cada caso de uso. Destacar que en el caso de uso 1 se ha decidido hacer un solo diagrama para todos los subcasos, pues todos necesitan básicamente las mismas funcionalidades. Lo mismo ocurre con el caso de uso 4 y 6.

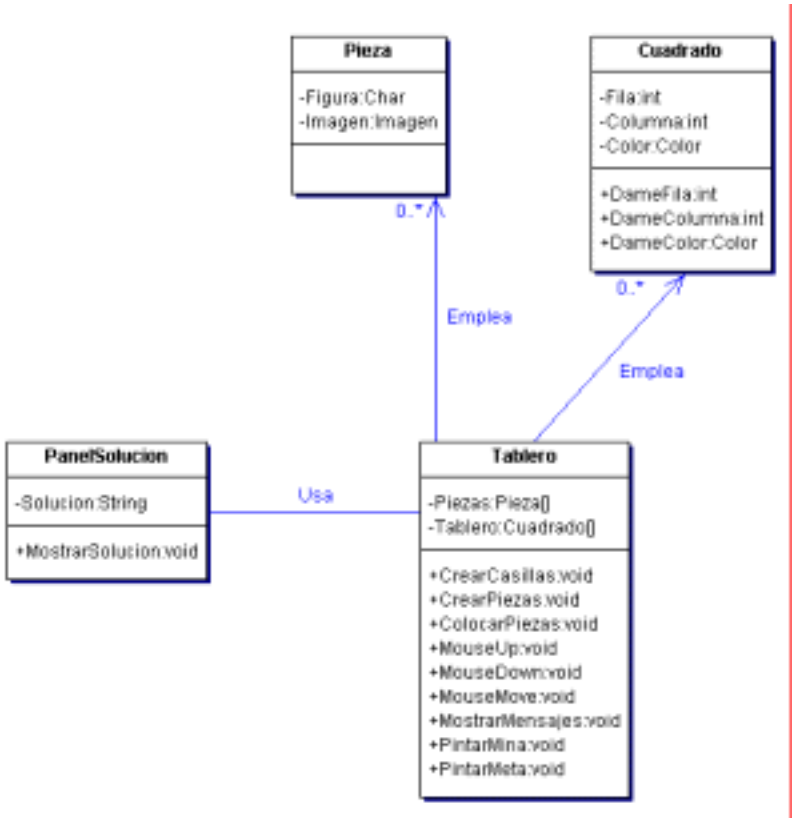
Caso de uso 1: Generar ejercicios



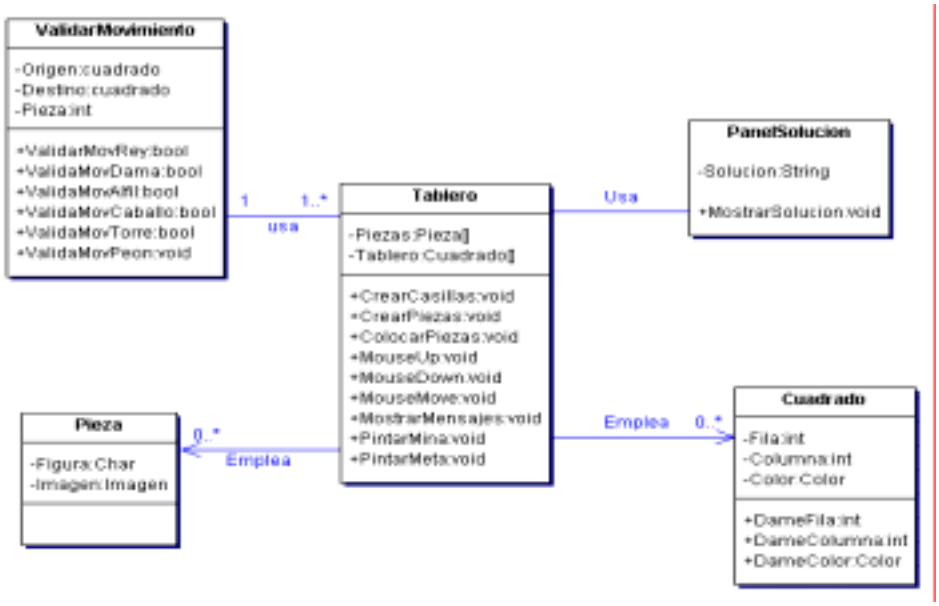
Caso de uso 2: Ejercicio 1



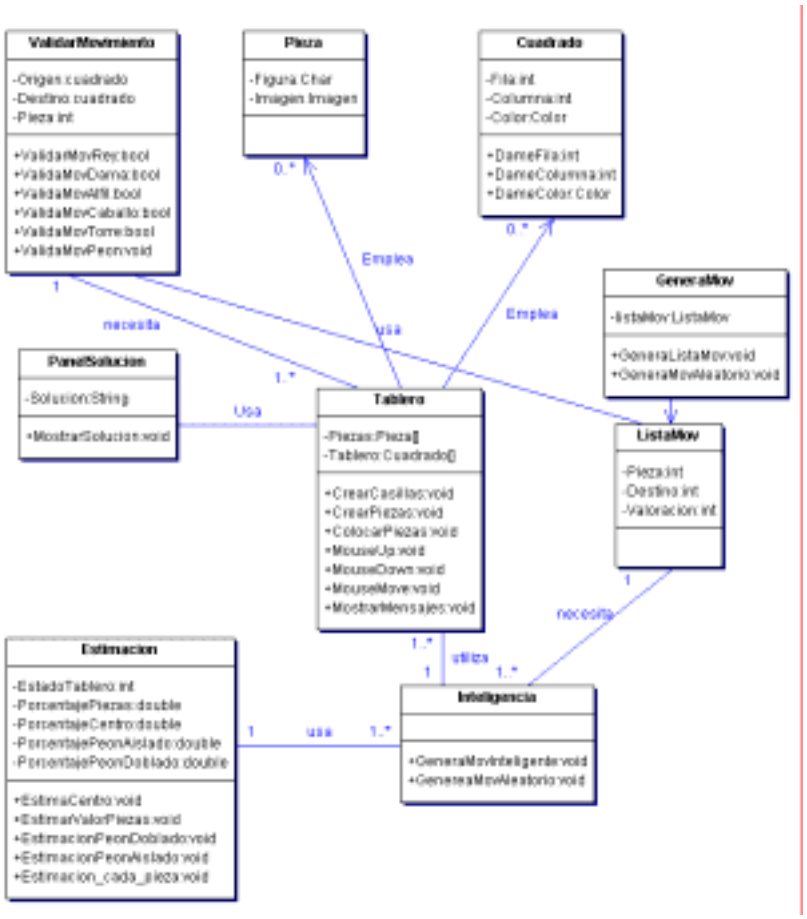
Caso de uso 3: Ejercicio 2



Caso de uso 4 y 6: Ejercicio 3 y 5



Caso de uso 5: Ejercicio 4



9. Implementación

A continuación se comentan los aspectos propios de la implementación del sistema. Se hará un estudio diferenciado de la herramienta generadora de ejercicios y de los ejercicios en sí mismos.

9.1 Ejercicios Applets

Para hacer un mejor seguimiento de la implementación se irá recorriendo las clases creadas en orden ascendente en la jerarquía.

Clase Cuadrado

La clase cuadrado es la encargada de implementar las casillas que conforman un tablero. Para ello dispone de una serie de atributos de entre los cuales lo más característicos son:

- **static final int lado:** Este atributo de clase representa la longitud del lado de cada cuadrado en píxeles. Se empleará para dibujar los cuadrados.
- **char fila:** De tipo carácter toma valores entre 1 y 8. Determina la fila a la que pertenece la casilla.
- **char columna:** De tipo carácter también, toma sus valores en el rango A .. H. Caracteriza la columna que ocupa el cuadrado dentro del tablero.
- **int numfil, numcol:** Son la representación numérica de los atributos Fila y Columna.
- **Container padre:** contiene una referencia al tablero que contiene esta casilla.
- **char color:** Puede ser 'b' para casillas blancas y 'n' para las negras.
- **int x, y:** Se refieren a la posición del extremo superior izquierdo del cuadrado dentro del tablero medido en píxeles. Así los valores de estos atributos para el cuadrado (8,A) – es decir el superior izquierdo – serán (0,0), para el cuadrado contiguo al anterior (lado, 0) y así sucesivamente
- **boolean marcada, es_sol:** Empleados para ejercicios del tipo 1, en los que es necesario saber si esa casilla ha sido marcada por el usuario y si pertenece realmente al conjunto de soluciones del problema.

- **boolean es_mina, es_meta:** Atributos para ejercicios de tipo 3, en donde debemos saber si el cuadrado contiene una mina, o si es la meta que debe alcanzar el usuario.

Los métodos más importantes de esta clase son:

Constructores de clase:

public **cuadrado**()

Constructora por defecto, inicializa todos los atributos no inicializados ya en la declaración a valores por defecto.

public **cuadrado**(int f, int c)

Construye un cuadrado actualizando los atributos numfil y numcol con los valores indicados por f y c respectivamente, y fila y columna con sus equivalentes, asignando al resto valores por defecto.

public **cuadrado** (Container acont, int ax, int ay, char afil, char acol, char acolor)

Constructora completa para el cuadrado. Recibe todos los valores de sus atributos y los asigna convenientemente.

public **cuadrado**(cuadrado that)

Constructora de copia. Devuelve un nuevo cuadrado como copia del que se le pasa como parámetro.

public void **copy**(cuadrado that)

Procedimiento para copiar un cuadrado que copia solo la información que relevante.

public boolean **isValid**()

Función que nos dice cuándo un cuadrado contiene valores correctos para los atributos fila y columna.

public void **reset**()

Actualiza los atributos de un cuadrado con sus valores por defecto.

public boolean **equals**(Object o)

Función que nos dice cuándo dos cuadrados son iguales.

public boolean **nula** ()

Nos dice cuándo un cuadrado tiene en sus atributos los valores que se asignan por defecto, es decir, cuándo el cuadrado es nulo.

public boolean **incluye** (int ax, int ay)

Dadas las coordenadas de un punto, devolverá cierto si éstas están contenidas dentro de los límites del cuadrado y falso en otro caso.

Clase Pieza

La clase pieza representa a las piezas que componen una partida de ajedrez. Los atributos que la caracterizan son, principalmente:

- **static final int ancho, alto:** Miden la longitud de la imagen de la pieza en píxeles.
- **Image imagen:** Representa la imagen de la pieza.
- **char color:** De tipo carácter puede ser 'W' representando el color blanco, o 'B' para las figuras negras.
- **char figura:** También de tipo carácter toma valores en el conjunto 'P','R','N','B','K','Q', que se corresponden con las piezas peón, torre, caballo, alfil, rey y reina.
- **String tipo:** Este atributo identifica unívocamente a cada pieza del tablero. Está formado por la unión del color y la figura.
- **cuadrado casilla:** Contiene una referencia al cuadrado en el que se encuentra la pieza. En el caso de que no esté en el tablero, la pieza va a parar a un cuadrado vacío.
- **Container padre:** Referencia al tablero sobre el que se dibujan las piezas.

Los métodos más significativos de esta clase son:

Constructora de clase:

public **pieza**(Container cont, char acolor, char afigura, Image dibujo, cuadrado acasilla)

Inicializa los atributos del objeto con los valores que se pasan como parámetros.

public void **mueve** (cuadrado sq)

Mueve la pieza de la casilla en la que está actualmente a la que se le pasa como parámetro.

public boolean **incluye** (int x, int y)
Dadas las coordenadas de un punto nos dice si éste está contenido en el contorno de la pieza.

public int **piezaToInt**()
Transforma una pieza en un entero para poder trabajar mejor con ella.

Clase Tablero

La clase tablero es la que guarda toda la información referente al tablero de ajedrez. Contiene las estructuras y los métodos necesarios para poder gestionar la ejecución de los cinco tipos de ejercicios de ajedrez. Hereda de la clase Panel para poder albergar las piezas y para poder dibujar sobre él los cuadrados y las minas –en ejercicios del tipo 3 –. Implementa además la interfaz Historia

Contiene atributos estáticos constantes que definen las posiciones de las piezas dentro de la estructura que las guarda.

Sus principales atributos son:

- **static final int borde:** Medida en píxeles que nos da la anchura del recuadro que rodea al tablero.
- **int uso:** Dentro del rango 1.. 5, identifica para qué tipo de ejercicio se está usando el tablero.
- **pieza[] piezas:** Contiene todas las piezas del juego.
- **cuadrado[] casillero:** Representa las casillas de todo el tablero.
- **Position posicion:** Indica la posición de una pieza dentro de un tablero.
- **cuadrado start:** Contiene el cuadrado con la pieza a mover en el caso de que se mueva una pieza de una casilla a otra del tablero.
- **cuadrado stop:** Representa al cuadrado al que se mueve la pieza.
- **int pieza_movida:** Contiene la posición de la pieza que se mueve dentro del array de piezas o –1 si no se ha movida ninguna pieza.
- **int casilla_marcada:** Contiene el índice del cuadrado sobre el que se pincha con el ratón dentro del array de casillas.
- **int m_toMove:** Representa a quién le toca mover pieza, si a las blancas o a las negras.
- **PanelPpal padre:** Referencia al contenedor del tablero.

- **int num_movimientos:** Va guardando el número de movimientos que hace el usuario sobre el tablero. Útil para ejercicios del tipo 3 y 5.
- **boolean fin:** Nos dice si se ha llegado al final del ejercicio.
- **String solucion:** Para ejercicios de tipo 1, guarda el conjunto de casillas que forman la solución correcta.
- **String minas:** Para ejercicios del tipo 3, guarda el conjunto de casillas que contienen minas.
- **Image imagen_meta:** Para ejercicios del tipo 3, guarda la imagen que representa la meta a alcanzar por el usuario.
- **int mov_jaque:** Para ejercicios del tipo 5, dice el número máximo de movimientos que se pueden hacer – sumando los del usuario y los de respuesta del computador – para que el primero le dé jaque al segundo.

Atributos que controlan el comportamiento inteligente del computador en el transcurso de una partida:

- **boolean think:** Si es cierto, el computador debe responder a los movimientos realizados por el usuario.
- **DatosNodo respuesta:** Estructura para guardar la repuesta del nodo del árbol que tienen mejor valoración, de los posibles movimientos siguiendo la técnica Minimax.
- **GeneraMov Aleatorio:** Genera un movimiento aleatorio de todos los posibles que se pueden realizar.
- **float ganar_material:** Dentro del rango 0..1. Cuanto más próximo esté su valor a 1, más tenderá el computador a elegir los movimientos que le hacen comer piezas.
- **float ganar_centro:** Dentro del rango 0..1. Cuanto más próximo esté su valor a 1, más tenderá el computador a elegir las jugadas que le hacen controlar el centro del tablero.
- **boolean es_aleatorio:** Si es cierto, el computador elegirá aleatoriamente una de las jugadas de entre las que puede hacer. En otro caso, seguirá la técnica Minimax para quedarse con la mejor de ellas.

Los métodos más destacados de esta clase son:

Constructoras de clase:

public **Tablero** (PanelPpal pad ,int tipo, Image[] imagen_piezas,
String inicial, String sol)

Construye un tablero para ejercicios del tipo 1. Inicializa los atributos del objeto con los valores que se le pasan como parámetros.

El parámetro 'inicial' contiene la configuración inicial de las piezas dentro del tablero, mientras que 'sol' guarda las casillas que forman parte de la solución del ejercicio. Imagen_piezas es un array que contiene todas la imagen de las piezas del tablero y que se utiliza en la construcción de las mismas.

public **Tablero** (PanelPpal pad, int tipo, Image[] imagen_piezas, String inicial)

Construye un tablero para ejercicios de tipo 2.

public **Tablero** (PanelPpal pad, int tipo, Image[] imagen_piezas,
Image imagen_met, String inicial, String aminas, String meta)

Construye un tablero para ejercicios de tipo 3.

Los valores 'aminas' y 'meta' guardan las casillas en donde están las minas y la meta, respectivamente. 'imagen_met' contiene la imagen que representa la meta a alcanzar por el usuario.

public **Tablero** (PanelPpal pad, int tipo, Image[] imagen_piezas,
String inicial, float porcentaje_ganar_material,
float porcentaje_ganar_centro, boolean aleat)

Construye un tablero para ejercicios de tipo 4.

Los atributos que guardan la información para controlar la respuesta del computador se actualizan con los valores 'porcentaje_ganar_material', 'porcentaje_ganar_centro' y 'aleat'.

public **Tablero** (PanelPpal pad, int tipo, Image[] imagen_piezas,
String inicial, int num_mov)

Construye un tablero para ejercicios de tipo 5.

El número de movimientos máximos permitidos en total se actualiza con el parámetro 'num_mov'.

Los métodos comunes para los cinco tipos de ejercicios son:

public void **crea_casillas** ()

Construye las 64 casillas que forman el tablero de ajedrez más una nula que se emplea para las piezas que no están en ninguna de las casillas.

public void **crea_piezas** (Image[] imagen_piezas)

Construye las 32 del juego recibiendo como parámetro una colección que contiene las imágenes de cada tipo de pieza.

public void **coloca_piezas** (String colocacion)

Coloca las piezas en las casillas que le corresponden según el parámetro 'colocación', que contiene una lista de todas las piezas que deben estar presentes en la configuración inicial del tablero junto con las casillas que éstas deben ocupar.

private int **strToPosCasilla** (String casilla)

A partir de un String con la fila y la columna de una casilla devuelve la posición que ocupa dentro del array casillero.

public int **posicion_pieza** (char color, char figura)

Devuelve la posición de una pieza dentro del array de piezas identificándola por el color y la figura que representa.

public void **paint** (Graphics g)

Procedimiento para pintar el tablero

public void **paint_casillas** (Graphics g)

Pinta los cuadrados que forman el casillero del tablero de ajedrez. Contiene además llamadas a procedimientos especiales en el caso de que haya que hacer alguna tarea extra a la hora de pintar los cuadrados en función del tipo de ejercicio que estemos implementando.

public void **paint_piezas** (Graphics g)

Carga las imágenes de las piezas en el tablero colocándolas en las casillas que les corresponden.

Public void **update** (Graphics g)

Similar a hacer un repaint del tablero. Contiene llamadas especiales para los distintos tipos de ejercicios.

public boolean **mouseDown** (Event e, int x, int y)

Gestiona el evento que se genera cuando el usuario pulsa el botón izquierdo del ratón. Identifica la pieza sobre la que se ha pulsado o, si ésta no existiese, la casilla sobre la que se produjo el evento. Contiene además llamadas a procedimientos especiales para gestionar este evento en los cinco tipos de ejercicios.

public boolean **mouseUp** (Event e, int x, int y)

Contiene llamadas a procedimientos que tratan el evento producido cuando el usuario levanta el dedo del botón izquierdo del ratón para los cinco tipos

de ejercicios.

public boolean **mouseMove** (Event e, int x, int y)

Gestiona el movimiento del ratón a través del tablero. Se utiliza principalmente para cambiar la apariencia del cursor cuando éste se coloca sobre una de las piezas del tablero.

Los métodos usados para gestionar el tablero para el primer tipo de ejercicios son:

public void **paint_casillas_uso1** (Graphics g)

Se encarga de marcar las casillas seleccionadas por el usuario como parte de la solución con una 'X' y las que verdaderamente son solución – cuando el usuario decide dar por finalizado el ejercicio – con un 'O'.

public void **mouseDown_uso1**()

Se encarga de desmarcar una casilla en el caso de que ésta estuviera previamente marcada y el usuario vuelva a apretar el botón del ratón sobre ella.

public boolean **marcar_casillas_especiales**()

Actualiza los valores 'es_sol' y 'es_mina' – para el tipo 3 – de las casillas y devuelve un boolean que será tomado en cuenta sólo si se usa para ejercicios de tipo 1 y que será cierto si el usuario marcó todas las casillas que forman la solución correcta del ejercicio, y sólo esas.

Para ejercicios del segundo tipo, no es necesario añadir ninguna funcionalidad nueva a las que básicas que incorpora el tablero para cualquiera de los otros tipos.

Para ejercicios de tipo 3 se requieren los siguientes métodos:

public void **paint_casillas_uso3** (Graphics g)

Pinta las minas a sortear y la meta a alcanzar como corresponda.

public void **update_uso3**(Graphics g)

Contiene una llamada al update_uso4, puesto que hace lo mismo.

public boolean **mouseUp_uso3**(Event e, int x, int y)

Se encarga de mover la pieza que ha de alcanzar la meta por el tablero, vigilando que los movimientos realizados sean correctos y que no se pase

por encima de ninguna de las minas, llevando además la cuenta del número de movimientos que ha realizado el aprendiz hasta alcanzar la meta.

public boolean **atraviesa_mina** (cuadrado destino)

Devuelve cierto si la pieza que está siendo en ese momento arrastrada atraviesa alguna mina en su camino desde el cuadrado en el que está hasta el cuadrado 'destino'.

Los métodos para ejercicios de tipo 4 y 5 son muy similares. Son los que se muestran a continuación:

public void **update_uso4**(Graphics g)

En caso de que alguna pieza haya cambiado de casilla, la pinta en la posición que ocupa actualmente.

public boolean **mouseUp_uso4** (Event e, int x, int y)

Es el encargado de llevar a cabo las tareas necesarias cuando se levanta el botón izquierdo del ratón. Si había una pieza que estaba siendo arrastrada, intenta moverla al cuadrado destino y genera una respuesta a este movimiento. Si el ejercicio es de tipo 5, ésta será aleatoria. Si es de tipo 4, podrá ser aleatoria o dirigida por el árbol de decisión Minimax, según haya seleccionado el usuario que haya creado el ejercicio.

public boolean **muevePieza**(int destino)

Verifica que la pieza que está siendo arrastrada en ese momento puede moverse a la casilla que ocupa la posición 'destino' dentro del casillero. Devuelve cierto, si el movimiento es válido y ha podido llevarse a cabo y falso en otro caso.

Además la clase Tablero implementa los métodos heredados de la interfaz 'Historia'.

Clase Panel Principal

Esta clase está pensada como contenedor de alto nivel que albergue el tablero de ajedrez, así como otros elementos que forman parte del ejercicio.

Los atributos de esta clase son:

- **Tablero tablero:** Que representa al tablero que va dentro del panel principal.

- **Panel panelmsj:** Panel que contiene información útil sobre el ejercicio situado en la parte inferior del panel principal.

Los métodos de esta clase están compuestos casi exclusivamente por constructoras. Así tenemos hasta 5, una para cada tipo de ejercicio:

public **PanelPpal** (int uso, Image[] imagen_piezas, URL finicial, URL fsolucion,
URL fmsjerror, URL fmsjexito, URL fmsjpregunta)

Construye un panel principal para albergar ejercicios del tipo 1. Recibe como parámetros el número que identifica el uso, un array que contiene las imágenes de las piezas del tablero, el fichero con la configuración inicial de las piezas, el fichero con las casillas que forman la solución correcta, el fichero que contiene el mensaje que se mostrará en caso de que se resuelva el ejercicio correctamente, el que contiene el mensaje que se mostrará en caso de que no se resuelva como es debido y el que guarda la pregunta que encabezará el ejercicio.

Se encarga de crear un tablero, de colocar la pregunta al principio del ejercicio y de añadir un panel solución.

Publio **PanelPpal** (int uso, Image[] imagen_piezas, URL finicial, URL fpregunta,
URL fopciones, URL frespuesta)

Construye un panel en el que irán ejercicios de tipo 2. Recibe para ello el número que identifica el uso, un array con las imágenes de las piezas, el fichero con la configuración inicial de las piezas, el fichero que guarda la pregunta que encabeza el ejercicio, el fichero con las posibles opciones y el que contiene la respuesta correcta.

Se encarga de crear un tablero, de colocar la pregunta al principio del ejercicio, de añadir un panel que albergue la respuesta y otro que contenga las posibles opciones.

public **PanelPpal** (int uso, Image[] imagen_piezas, Image imagen_meta,
URL finicial, URL fminas, URL fmeta)

Construye un panel que contendrá ejercicios del tipo 3. Recibe como parámetros el número que identifica el uso, el array con las imágenes de las piezas, la imagen que identifica la meta a alcanzar por el aprendiz, y los ficheros con las configuraciones de las piezas, las minas y la meta.

Se encarga de crear un tablero, de colocar la pregunta al principio del ejercicio y de añadir un panel que vaya mostrando el número de movimientos realizados.

public **PanelPpal** (int uso, Image[] imagen_piezas, URL finicial, URL fmsj,
float porcentaje_ganar_material,
float porcentaje_ganar_centro, boolean aleatorio)

Construye un panel que contendrá ejercicios de tipo 4. Recibe como parámetros el número que identifica el uso, el array con las imágenes de las piezas, el fichero con la configuración inicial de las piezas, un fichero que contendrá un mensaje que encabece la partida, el peso que debe dar el computador a las jugadas que ganan material, el que se dará a las que controlan el centro del tablero y si la elección de jugadas es aleatoria o no.

Es el encargado de crear un tablero, de colocar la pregunta al principio del ejercicio y de añadir un panel que irá mostrando mensajes en el transcurso del juego.

public **PanelPpal** (int uso, Image[] imagen_piezas, URL finicial,
URL fmsj, int num_movimientos)

Construye un panel para ejercicios de tipo 5. Recibe como parámetros el número que identifica el uso, el array con las imágenes de las piezas, el fichero con la configuración inicial de las piezas, el mensaje que encabezará el ejercicio y el número de movimientos en total (entre el aprendiz y el computador) que se pueden realizar en el mismo.

Creará un tablero, colocará la pregunta al principio del ejercicio y añadirá un panel que se irán mostrando mensajes en el transcurso del ejercicio.

Esta clase contiene además un método que no será de utilidad en varios casos:

public String **fileToStr** (URL archivo)

Lee el contenido de un fichero y lo devuelve en una cadena de caracteres.

Clase Panel Información

Esta clase está pensada como panel auxiliar para albergar cualquier tipo de mensajes que nos puedan ser útiles en el transcurso de los ejercicios. Se emplea para los Applets de los tipos 3, 4 y 5.

Los atributos que contiene son:

- **PanelPpal padre:** referencia al panel que lo contiene.
- **TextArea txt:** zona de texto para mostrar los mensajes.

El único método que contiene es el constructor:

public **PanelInfo** (PanelPpal pad)

Recibe una referencia al panel que lo contiene. Crea la zona de texto inicializándola con un mensaje por defecto que podrá ser modificado en el transcurso de los ejercicios.

Clase Panel Botones

Clase que alberga los botones que le permiten al aprendiz seleccionar una de entre las distintas opciones a una respuesta en ejercicios de tipo 2.

Contiene el siguiente atributo:

- **PanelPpal padre:** referencia al panel padre que lo contiene

El único método que tiene es la constructora de clase:

public **PanelBotones** (PanelPpal pad, String opciones)

Construye el panel recibiendo como parámetros una referencia al panel padre y un string con las posibles opciones a una pregunta.

Clase Panel Respuesta

Esta clase contiene la información extra que requieren los ejercicios de tipo 2, en los que se ha de mostrar cuál es la respuesta correcta a una pregunta planteada.

Contiene los siguientes atributos:

- **Button bsolucion:** botón que deberá pulsar el aprendiz cuando quiera saber la respuesta correcta.
- **PanelPpal padre:** Referencia al panel que lo contiene.
- **TextArea txt:** Zona de texto en el que mostrar la solución correcta.
- **String msjsol:** Texto con la solución correcta al ejercicio.

El único método que contiene es la constructora de clase:

public **PanelRespuesta** (PanelPpal pad, String respuesta)

Construye el panel a partir de una referencia al panel padre y con la respuesta correcta al ejercicio.

Clase Panel Solución

Clase creada para ejercicios de tipo 1. Contiene los elementos necesarios para hacer visible la respuesta correcta cuando el aprendiz desea dar por finalizado el ejercicio.

Los atributos que forman parte de esta clase son:

- **Button solucion:** Botón para dar por acabado el ejercicio.
- **PanelPpal padre:** Referencia al panel que lo contiene.
- **TextArea txt:** Zona de texto en el que mostrar los mensajes.
- **String error:** Mensaje que se muestra cuando el aprendiz no ha resuelto el ejercicio correctamente.
- **String exito:** Mensaje que se muestra cuando el aprendiz resuelve el ejercicio correctamente.

El único método es la constructora de clase:

public **PanelSolucion** (PanelPpal pad, String msjerror, String msjexito)

Construye el panel solución tomando como parámetros la referencia al panel padre, el mensaje que se dará en caso de error, y el que se mostrará al aprendiz en caso de que resuelva adecuadamente el ejercicio.

Clase Estimación

La clase estimación es la encargada de valorar cada jugada teniendo en cuenta distintos parámetros, como: la suma de todos los valores de las piezas, la movilidad de los caballos, torres, alfiles, dama, el dominio del centro del tablero por el peón y la posibilidad del peón de comer una pieza contraria. Cada uno de los parámetros anteriores se valora con unos porcentajes determinados que nos indican la importancia que se le da a cada cosa. El valor de estos parámetros en un rango de 0 a 0.7 es la siguiente:

- **Por_piezas**: Porcentaje que estima la suma total del valor de las piezas multiplicada por un valor base 100 => 0.7
- **Por_peon**: Porcentaje que estima el dominio del centro del tablero => 0.6
- **Por_alfil**: Porcentaje que estima los movimientos de los alfiles => 0.1

- **Por_torre**: Porcentaje que estima los movimientos de las torres => 0.1
- **Por_caballo**: Porcentaje que estima los movimientos de los caballos => 0.1
- **Por_dama**: Porcentaje que estima los movimientos de la dama => 0.5
- **Por_peoncome**: Porcentaje que estima los movimientos donde un caballo come a una pieza contraria => 0.1

A continuación se muestra la estimación detallada para cada atributo:

ValorPiezas:

Calcula la suma del valor de todas las piezas, multiplicado por un valor base (100). El valor de cada pieza quedaría:

Peón: 1 * valor_base
Caballo: 3 * valor_base
Alfil: 3.5 * valor_base
Torre : 5 * valor_base
Dama: 9 * valor_base

ValorPeon:

Se valora el dominio del centro del tablero por los peones.
Para los cuatro cuadrados centrales:
Si no hay ninguna pieza nuestra:
Si el cuadrado es atacado por una pieza contraria: -100
Si ese cuadrado esta protegido por una pieza tuya: 30
Si el movimiento es posible: 15
Comprobar si puede comer a alguna contraria dependiendo de cual sea se suma un valor u otro
Si hay pieza nuestra:
Si el cuadrado es atacado por una pieza contraria: -100
Si ese cuadrado esta protegido por una pieza tuya: 30

ValorTorre:

Se valora la posibilidad de movimiento de las torres, teniendo en cuenta el efecto sobre el contrario:

Para las posiciones a las que se puede mover una torre teniendo en cuenta la posición actual de la misma:

Si no hay ninguna pieza nuestra:

Si el cuadrado al que se puede mover la torre es atacado por una pieza contraria –100

Si el cuadrado al que se mueve está protegido por una pieza tuya 110

Si es un movimiento posible 1

Comprobar si puede comer a alguna contraria dependiendo de cual sea se suma un valor u otro

Si hay pieza nuestra:

Si el cuadrado al que se puede mover la torre es atacado por una pieza contraria –100

Si el cuadrado al que se mueve está protegido por una pieza tuya 110

ValorAlfil:

Se valora la posibilidad de movimiento de los alfiles, teniendo en cuenta el efecto sobre el contrario:

Para las posiciones a las que se puede mover un alfil teniendo en cuenta la posición actual del mismo:

Si no hay ninguna pieza nuestra:

Si el cuadrado al que se mueve es atacado por una pieza contraria: –150

Si el cuadrado al que se mueve está protegido por una pieza tuya: 180

Si es un movimiento posible +1

Comprobar si puede comer a alguna contraria dependiendo de cual sea se suma un valor u otro

Si hay pieza nuestra:

Si el cuadrado al que se mueve es atacado por una pieza contraria: –150

Si el cuadrado al que se mueve está protegido por una pieza tuya: 180

ValorDama

Valoramos la posibilidad de movimiento de la dama, teniendo en cuenta el efecto sobre el contrario:

Para las posiciones a las que se puede mover la dama teniendo en cuenta la posición actual de la misma:

Si no hay ninguna pieza nuestra:

Si el cuadrado al que se mueve es atacado por una pieza contraria –200

Si el cuadrado al que se mueve está protegido por una pieza contraria 300

Si es un movimiento posible: +1, +2, +3, +4

Comprobar si puede comer a alguna contraria dependiendo de cual sea se suma un valor u otro

Si hay pieza nuestra:

Si el cuadrado al que se mueve es atacado por una pieza contraria –200

Si el cuadrado al que se mueve esta protegido por una pieza contraria 300

ValorCaballo:

Se valora la posibilidad de movimiento de los caballos, teniendo en cuenta el efecto sobre el contrario:

Para las posiciones a las que se puede mover el caballo teniendo en cuenta la posición actual de la misma:

Si no hay ninguna pieza nuestra:

Si el cuadrado al que se mueve es atacado por una pieza contraria –140

Si el cuadrado al que se mueve esta protegido por una pieza contraria 170

Si es un movimiento posible: +1, +2, +3, +4

Comprobar si puede comer a alguna contraria dependiendo de cual sea se suma un valor u otro

Si hay pieza nuestra:

Si el cuadrado al que se mueve es atacado por una pieza contraria –140

Si el cuadrado al que se mueve esta protegido por una pieza contraria 170

Posteriormente se va a explicar la implementación de la clase detallando los atributos y los métodos más significativos y relevantes:

A continuación se muestran los atributos de esta clase:

- **Position posición**: Indica la posición de una pieza dentro de un tablero.
- **cuadrado [] casillero**: Representa las casillas de todo el tablero.
- **pieza[] piezas**: Representa a todas las piezas.

- **cuadrado start**: Representa la posición inicial de una pieza.
- **int Turno**: Indica si el turno es de las blancas o de las negras según el valor.
- **double valor_piezas**: Suma de todos los valores de las piezas de un color que hayen el tablero
- **double mov_alfiles**: Valor teniendo en cuenta la movilidad de los alfiles.
- **double mov_torres**: Valor teniendo en cuenta la movilidad de las torres
- **double peon_centro**: Valor teniendo en cuenta el dominio del centro del tablero
- **double peon_comiendo**: Valor teniendo en cuenta la posibilidad de que un peón coma
- **double mov_caballo**: Valor teniendo en cuenta la movilidad de los caballos
- **double mov_dama**: Valor teniendo en cuenta la movilidad de la dama
- **float porcentaje_piezas**: Porcentaje de importancia del valor de las piezas
- **float porcentaje_alfil**: Porcentaje de importancia de la movilidad de los alfiles
- **float porcentaje_torre**: Porcentaje de importancia de la movilidad de las torres
- **float porcentaje_peon**: Porcentaje de importancia del dominio del centro del tablero
- **float porcentaje_peonComiendo**: Porcentaje de la importancia de que un peón coma.
- **float porcentaje_caballo**: Porcentaje de importancia de la movilidad de los caballos
- **float porcentaje_dama**: Porcentaje de importancia de la movilidad de la dama
- **double estimacion_total**: Valor de la estimación total teniendo en cuenta todas las características estimadas.

En las siguientes líneas se explican todas las funciones detalladamente:

Constructor de la clase:

```
public Estimacion (Position pos, cuadrado[] cas, pieza[] piez, int turno,
                   float por_piezas, float por_alfil, float por_torre,
                   float por_peon, float por_peoncome, float por_caballo,
                   float por_dama)
```

Inicializa todos los atributos de la clase con los parámetros que se le pasan. Éstos se explican a continuación:

Pos: Representa una posición en el tablero

Cas: Representa una casilla del tablero

Piez: Representa a todas las piezas

Turno: Indica el tuno del jugador al que le toca mover

por_piezas: Porcentaje con el que valoramos la jugada de ganar material

por_alfil: Porcentaje con el que valoramos las jugadas de los alfiles

por_torre: Porcentaje con el que valoramos las jugadas de las torres.

por_peon: Porcentaje con el que valoramos si un peón está en el centro

por_peoncome: Porcentaje con el que se valora las jugadas donde el peón puede comer a una pieza

por_caballo: Porcentaje con el que valoramos las jugadas del caballo

por_dama: Porcentaje con el que valoramos la posición de la dama

public double **estimacion_total**()

Devuelve el valor resultante de la estimación total teniendo en cuenta cada valoración con un porcentaje determinado.

public void **modificar_ValorPeon**()

Calcula la estimación teniendo en cuenta el dominio del centro del tablero por el peón.

public void **modificar_ValPeonComiendo**(int ident)

Modifica el atributo peon_comiendo, del peón que se le pasa como atributo, teniendo en cuenta las posibilidades que tiene ese peón de comer a una pieza contraria.

public void **modificar_ValTorre**(int numTorre)

Modifica el atributo mov_torres, teniendo en cuenta la movilidad de las torres.

public void **modificar_ValAlfil**(int numAlfil)

Modifica el atributo mov_alfiles, teniendo en cuenta la movilidad de los alfiles.

public void **modificar_ValDama**(int numDama)

Modifica el atributo mov_dama, teniendo en cuenta la movilidad de la dama.

public void **modificar_ValCaballo**(int numCaballo)

Modifica el atributo mov_caballo, teniendo en cuenta la movilidad del caballo.

public Vector **estaATiroAlfil** (cuadrado cuadro)

Calcula todas las posibles casillas a las que se puede mover el alfil (Casilla_destino), teniendo en cuenta la posición en la que se sitúa actualmente (Casilla_actual). Dichas casillas se calculan de la siguiente manera:

Casilla_actual -> (x, y)

Casilla_destino-> (x', y')

Condición para que un alfil que está en *Casilla_ actual* se pueda mover a la *Casilla_destino*: |x-x'| = |y-y'|. Todas las casillas que cumplan dicha condición serán guardadas en el vector resultante como posibles movimientos del alfil que se sitúa en *Casilla_actual*.

public Vector **estaATiroTorre** (cuadrado cuadro)

Calcula todas las posibles casillas a las que se puede mover la torre (Casilla_destino), teniendo en cuenta la posición en la que se sitúa actualmente (Casilla_actual). Dichas casillas se calculan de la siguiente manera:

Casilla_actual -> (x, y)

Casilla_destino-> (x', y')

Condición para que una torre que está en *Casilla_ actual* se pueda mover a la *Casilla_destino*: x=x' o y=y'. Todas las casillas que cumplan dicha condición serán guardadas en el vector resultante como posibles movimientos de la torre que se sitúa en *Casilla_actual*.

public Vector **estaATiroPeonComiendo** (cuadrado cuadro)

Calcula las posibles posiciones a las que se puede mover peón para comer una pieza contraria (Casilla_destino), teniendo en cuenta la posición en la que se sitúa actualmente (Casilla_actual). Dichas casillas se calculan de la siguiente manera:

Casilla_actual -> (x, y)

Casilla_destino-> (x', y')

Condición para que un peón que está en *Casilla_ actual* se pueda mover a la *Casilla_destino*: (|x'-x|= 1) && (|y-y'|= 1). Todas las casillas que

cumplan dicha condición serán guardadas en el vector resultante como posibles movimientos del alfil que se sitúa en *Casilla_actual*.

public Vector **estaATiroDama** (cuadrado cuadro)

Calcula las posibles posiciones a las que se puede mover la dama (Casilla_destino), teniendo en cuenta la posición en la que se sitúa actualmente (Casilla_actual). Dichas posiciones son las mismas que para una torre y un caballo, con lo cuál lo único que hay que hacer es calcular las que están a tiro de torre y las que están a tiro de alfil.

public Vector **estaATiroCaballo** (cuadrado cuadro)

Calcula las posibles posiciones a las que se puede mover un caballo (Casilla_destino), teniendo en cuenta la posición en la que se sitúa actualmente (Casilla_actual). Dichas casillas se calculan de la siguiente manera:

Casilla_actual -> (x, y)

Casilla_destino-> (x', y')

Dx = |x-x'|

Dy = |y-y'|

Condición para que un caballo que está en *Casilla_ actual* se pueda mover a *Casilla_destino*: $dx^2 + dy^2 = 5$. Todas las casillas que cumplan dicha condición serán guardadas en el vector resultante como posibles movimientos del caballo que se sitúa en *Casilla_actual*.

Para realizar esta parte del proyecto nos ha sido muy útil información encontrada en Internet:

http: // www.geocities.com/zodiamoon/amyan/comofuncinonan.HTML ,

En está página se dan posibles valores para la estimación que nos han servido de guía. Esto junto con otros programas de ajedrez (SimpleChess) que están en la Web nos ha servido de orientación en la difícil tarea de la estimación.

Clase Historia

Esta clase se trata en realidad de una interfaz, necesaria para poder comprobar los movimientos. La interfaz se implementa en todas aquellas clases en la que es necesaria comprobar movimientos. En ella se definen los siguientes métodos:

public int **toMove()**
Necesario para saber quién mueve en cada jugada.

public boolean **enJaque()**
Comprueba si el rey está en jaque

public boolean **reyMovido**(int color)
Determina si se ha movido un rey en el pasado del color especificado en el parámetro *color*. Esta información es importante para saber si se puede realizar o no el enroque.

public boolean **saltoPeon**(cuadrado loc)
Este método se encarga de saber si un peón ha movido dos posiciones al frente en un solo movimiento. Este dato es necesario para saber si se puede comer al paso o no.

Clase Position

La clase Position se encarga de guardar y mantener la información necesaria para conocer el estado del tablero. Esto se traduce en la existencia de un atributo llamado ***_position*** que guarda la pieza que hay en cada posición del tablero por medio de una matriz bidimensional de enteros, de tal forma que en el caso de que haya pieza se almacena el número de la misma y en caso contrario un *–1.*

Cada pieza (independientemente del color) tiene asociado una constante en la clase de tipo int y, a su vez, existen otras dos constantes de tipo int para cada color. El tipo de pieza (incluyendo el color) se obtiene sumando las dos constantes antes citadas.

En esta clase, aparte de un constructor, un constructor de copia y un método que inicializa todas las posiciones de *_position* a –1 (sin pieza) existen dos métodos más con las siguientes funcionalidades:

1. Devolver la pieza que hay en una cuadrado determinado del tablero. Este método de declara como: public int getPiece(cuadrado loc)
2. Mete una pieza determinada (int) en un cuadrado determinado del tablero. Este método se llama: public void setPiece(cuadrado loc, int piece)

Clase Validar

Esta clase se encarga de comprobar que los movimientos de las piezas que se realizan en el juego están bien hechos, es decir, se ajustan a las reglas del juego del ajedrez. La información que se emplea para poder realizar esta comprobación es (aparecen en la clase en forma de atributos):

- **Position m_original**: Posición original y final tras los cálculos. Al ser de tipo Position lo que contiene es el estado del tablero.
- **Position m_position**: Donde se realizan todos los cálculos de validar. En un principio se trata de una copia de m_original.
- **Historia m_historia**: Se necesita para poder utilizar los métodos de la interfaz anteriormente comentados.
- **cuadrado m_start**: Cuadrado origen de la pieza que se va a mover. Es el *desde* del movimiento a validar.
- **cuadrado m_stop**: Cuadrado destino de la pieza a mover. Es el *hacia* del movimiento a validar.

Además de estos atributos existen otros para llevar la cuenta de si está comiendo, se está enrocando o se está comiendo al paso.

Aparte del constructor (donde se realiza una copia del estado del tablero que pasamos como parámetro) tenemos como método "principal" de esta clase:

public boolean **validaMovimiento**(cuadrado start, cuadrado stop, Historia historia) que es el que empieza con la validación del movimiento propiamente dicho. Este método se encarga de dos cosas; por un lado comprueba que los cuadrados *start* y *stop* son válidos, es decir, que ni son el mismo (nos estamos intentando mover al mismo sitio), y que contienen coordenadas dentro del tablero. Una vez realizada esta comprobación, lo siguiente es pasar a comprobar las reglas del movimiento siempre y cuando se verifique que no se está moviendo a una posición que provoca jaque al rey del jugador que está moviendo en ese momento.

El siguiente paso se realiza en el método validarReglas(). Lo que hace es averiguar que pieza se está intentando mover(usando las constantes definidas de las piezas). Según la pieza que sea, se tendrá que comprobar las siguientes reglas:

- Torre: orthogonal()
- Caballo: octagonal()

- Alfil: diagonal()
- Dama: orthogonal() y diagonal()
- Rey: adjacent() y checkCastle();
- Peon: pawnMove(), pawnCapture(), comerAlpaso()

A continuación comentamos a grandes rasgos lo que hace cada uno de estos métodos:

Orthogonal: en este método se comprueba que se está realizando un movimiento ortogonal, es decir, se está moviendo en horizontal o en vertical. Esto se comprueba restando las posiciones finales e iniciales tanto en filas como en columnas, y comprobando que una de las dos es cero. Además, una vez comprobado que se trata de un movimiento ortogonal es necesario comprobar que en el recorrido del movimiento no hay ninguna pieza entre medias.

Octagonal: se comprueba que se está ante un movimiento válido para el caballo. Esto se verifica si las restas en valor absoluto de la fila final menos la inicial y la columna final menos la inicial dan como resultado uno y dos o dos y uno respectivamente.

Diagonal: para este caso lo que se necesita comprobar es que las restas en valor absoluto de la fila final menos la inicial y la columna final menos la inicial dan el mismo resultado. Además, como ocurría con el movimiento ortogonal, es necesario comprobar que en cada casilla que conforma el recorrido del movimiento no hay ninguna pieza.

Adjacent: para comprobar que se trata de un movimiento adyacente únicamente hay que verificar que la resta en valor absoluto de la fila final menos la inicial y la columna final menos la inicial dan como resultado cero o uno.

CheckCastle: con este método se comprueba el enroque. Por un lado hay que comprobar si se trata de un enroque corto o largo teniendo en cuenta para ello el rey que se está moviendo y la columna a la que se quiere mover.. Para el caso del enroque corto es necesario comprobar que las piezas intermedias entre el rey y la torre están vacías. Además, usando los métodos de la interfaz historia, hay que asegurarse que el rey no se ha movido en el pasado y que no está siendo atacado por alguna pieza. Una vez satisfechas estas condiciones es necesario tener en cuenta que no sólo se moverá el rey sino que también se moverá la torre según el tipo de enroque y el turno del que juega.

Para el enroque largo tenemos lo mismo que en el caso anterior, tan sólo la posición de la torre cambia. Hay que destacar que una vez realizadas las comprobaciones hay que actualizar los atributos booleanos destinados a comprobar si se ha realizado el enroque o no, que serán usados posteriormente en la clase Tablero a la hora de realizar el movimiento de la pieza.

pawnMove: en este caso se realizan varias comprobaciones. Por un lado se comprueba que se está realizando un movimiento vertical y, según muevas blancas o negras, la diferencia entre la fila final e inicial puede ser uno o dos. En el caso de que la diferencia sea uno, hay que asegurarse de que no hay una pieza en el destino del movimiento (accediendo a _position), pues el peón no puede comer de esta manera. En el caso de que la diferencia sea dos, no solo hay que comprobar que en el destino no haya ninguna pieza, sino que también hay que asegurarse de que no hay pieza entre en el recorrido del movimiento (una sola casilla en este caso) y, además, de que se trata de un peón de la segunda o tercera fila en el caso de las blancas, o séptima y sexta fila en el caso de las negras.

PawnCapture: para comprobar que la captura del peón es correcta tan sólo hay que verificar que se trata de un movimiento en diagonal de una sola casilla (hay que tener en cuenta si se trata de blancas o negras) y que en el destino hay una pieza del contrario.

ComerAlpaso: en este método se verifica las reglas del caso en el que se come al paso. Por un lado hay que comprobar que se está tratando peones que están en la cuarta fila (en el caso de que jueguen blancas), o fila quinta (caso de las negras). Una vez realizada esta comprobación y que la pieza que se va a capturar es peón y movió anteriormente (existe un método destinado a tal efecto en Historia) se trata de actualizar el estado del tablero eliminando el peón comido por el nuevo, y actualizando los atributos de comer al paso que existen en esta clase. Estos serán usando por la clase Tablero a la hora de mover la pieza.

Por último, destacar que en la clase hay un método que se llama **pieceMove** que previene de capturarse a sí mismo y, además, comprueba el hecho de que se esté comiendo una pieza, es decir, que en la posición a la que se esté moviendo haya una pieza del contrario. En todo caso, en este método siempre se acaba actualizando la casilla que era origen del movimiento a –1 (sin pieza).

Clase Datos Nodo

Esta clase se emplea como tipo del nodo del árbol que se forma al realizar el alfa-beta. Tiene tres atributos enteros:

- **Valoracion**: valoración de la jugada
- **Pieza**: pieza que se ha movido en la jugada
- **Destino**: lugar a dónde se ha movido la pieza.

Ninguno de los métodos que dispone esta clase tiene alguna particularidad, tratándose de métodos que devuelven y actualizan el valor de los atributos de la clase.

Clase GeneraMov

Esta clase se encarga de generar una lista de tipo ListaMov de los movimientos que se pueden realizar para un jugador dado y un estado determinado del tablero. Tiene un método *principal* que realiza el trabajo importante; recibe el nombre de GeneraListaMov. Es este método el que realmente genera la lista de movimientos. Se consigue comprobando todos los posibles movimientos que puede hacer un determinado jugador; a través de un bucle, se coge cada pieza del jugador y se recorre todo el tablero intentando validar el movimiento. En el caso de que el movimiento sea correcto se añade a la lista. Hay que destacar que es necesario llevar una copia auxiliar del estado del tablero (Position auxPos) para poder deshacer los movimientos.

Por otro lado, existe un método llamado generaMovAleatorio() que se encarga de situar la lista de movimientos generada en una posición aleatoria. El mismo método se encarga de generar un número aleatorio (usando la clase java.lang.Math) entre cero y el tamaño de la lista. De esta forma se consigue que todos los movimientos tengan la misma probabilidad de ser elegidos. Este método es necesario para el caso en el que se desee jugar contra un oponente que juega de forma aleatoria.

Clase ListaMov

Esta clase se encarga de implementar las operaciones habituales de una lista doblemente enlazada.

La información que se guarda en cada nodo es la siguiente:

- **protected int pieza**: pieza que se va a mover
- **protected int destino**: destino de la pieza a mover
- **protected double valoracion**: valoración obtenida con este movimiento

Clase Minimax

En esta clase se implementa el algoritmo Minimax con poda alfa-beta. Para esta clase se necesita toda la información necesaria para poder "jugar" al ajedrez que conformarán los atributos de la clase:

- **private int m_toMove**: Turno del que juega: 8 blancas, 0 negras
- **private Position posicion = new Position():** Estado del tablero
- **public pieza[] piezas**: Piezas
- **private DatosNodo Respuesta = new DatosNodo()**: Contendrá la información que hay en el nodo del árbol
- **private cuadrado[] casillero**: Casillas del tablero

El método que realiza propiamente el Minimax con poda alfa-beta se llama double **hazMiniMax**(int profundidad, double alfa, double beta, int turno) que como se puede ver, devuelve la estimación de la jugada. La implementación del algoritmo sigue la estructura clásica:

1. Se generan una lista con los movimientos posibles
2. Si se entra en el algoritmo siendo un nodo hoja o la lista de movimientos es vacía, se estima la jugada.
3. En otro caso se recorre la lista de movimientos generada previamente y se llama de forma recursiva a hazMiniMax pero con signo contrario y cambiando el turno del jugador. El valor obtenido de esta llamada se mete en un variable llamada *nuevoValor*
4. Si *nuevoValor* es mayor que alfa entonces se actualiza alfa con *nuevoValor*
5. Si alfa es mayor o igual que beta entonces se puede podar y no es necesario visitar más nodos (instrucción *break*)

Es importante señalar que para poder hacer los movimientos y tal y como está la estructura del programa, es necesario llamar al método ValidarMov(). Además, para poder deshacer los movimientos, se tiene que

usar una variable *auxPos* que servirá para copiar temporalmente el estado del tablero.

Clase Applet de Ajedrez

Esta es la clase que creará finalmente el Applet que irá insertado dentro de la página HTML. Para ello cuenta con los siguientes atributos:

- **PanelPpal mainPanel:** Referencia al panel que contiene todos los elementos que configuran el Applet ejercicio.
- **Image [] imagen_piezas:** Array que guarda las imágenes de las piezas del tablero.
- **int uso:** Variable que identifica el tipo de ejercicio que estamos creando.

Los métodos más significativos de esta clase son:

private void **imagenes_piezas** ()

Carga las imágenes de las piezas y las guarda en la variable destinada a tal efecto.

private Image **imagen_meta** ()

Carga la imagen que representará la meta a alcanzar por el usuario en ejercicios del tipo 3 y la devuelve como resultado.

public void **init** ()

Procedimiento de inicialización del Applet que extrae de la página HTML en la que va inmersa el Applet el tipo de ejercicio que se quiere crear y que contiene llamadas a los métodos que crean los paneles principales en función del tipo deseado.

private void **constructor_uso1**()

Construye el panel que albergará ejercicios de tipo 1.

private void **constructor_uso2**()

Construye el panel que albergará ejercicios de tipo 2.

private void **constructor_uso3**()

Construye el panel que albergará ejercicios de tipo 3.

private void **constructor_uso4**()

Construye el panel que albergará ejercicios de tipo 4.

private void **constructor_uso5**()
Construye el panel que albergará ejercicios de tipo 5.

public static void **main** (String args[])
Procedimiento principal para crear e inicializar el Applet.

9.2 Herramienta Generadora de ejercicios

La herramienta generadora de ejercicios escrita en C++ está compuesta principalmente de los formularios que conforman la interfaz gráfica de usuario. Debido al poco interés que presenta el código desde el punto de vista algorítmico nos limitaremos a comentar las clases más importantes que intervienen y sus funcionalidades principales.

Clase Tpiezas

Creada con el propósito de modelizar a las piezas que el usuario debe colocar en el tablero, contiene atributos sobre las coordenadas de la pieza, la imagen y el tipo y métodos para construir las piezas, para moverla de su posición actual a otra con coordenadas x e y, y para saber si un punto dado por sus coordenadas está dentro del perímetro de la pieza.

Clase ToStr

Pensada como una especie de librería auxiliar, contiene atributos y métodos estáticos que nos permiten pasar las casillas del tablero y las piezas a cadenas de caracteres y viceversa.

Clase TFormPru

Clase que implementa el formulario que contiene las imágenes que se muestran cuando el usuario pulsa el botón 'Vista preliminar' en la pantalla en la que se decide el tipo de ejercicio a generar.

Clase TFormDesc

Clase que implementa el formulario que contiene la información que se visualiza por pantalla cuando el usuario pulsa el botón 'Descripción' en la pantalla en la que se decide el tipo de ejercicio a generar.

Clase TFormIni

Implementa el formulario que se muestra cuando el usuario entra a la aplicación. Contiene atributos para almacenar la información que se requiere para la generación de la página HTML. Este es el único formulario autogenerado por la aplicación y que permanece con vida durante todo el tiempo que ésta está en funcionamiento.

Clase TFormDirectorio

Implementa el formulario en el que el usuario elige el directorio sobre el que se vuelcan los archivos que contienen la información del Applet.

Clase TFormElec

Implementa el formulario en el que el usuario deberá elegir uno de los cinco posibles tipos de ejercicios a generar. Contiene llamadas a las clases TFormPru y TFormDesc en los botones 'Vista preliminar' y 'Descripción', respectivamente.

Clase TForm1

Es la más importante de todas las clases. Contiene una componente del tipo TChessBoard1 que representa al tablero sobre el que se colocan las piezas. Tiene métodos para crear las figuras (piezas, minas y meta) abrir y guardar configuraciones (de piezas, minas y meta), generar el fichero solución para ejercicios de tipo 1, verificar que las configuraciones cumplen con los requisitos mínimos en función del tipo de ejercicio (que estén los reyes, que no haya piezas negras…), para el tratamiento de eventos de ratón sobre las figuras...

Clase TFormMovimientos

Crea el formulario que recoge la información que completa los datos del ejercicio 1: la pregunta que encabeza el ejercicio, el mensaje en caso de éxito y el mensaje en caso de error.

Clase TFormPregunta

Crea el formulario que recoge la información que completa los datos del ejercicio 2: la pregunta que encabeza el ejercicio, las opciones de respuesta y la respuesta correcta.

Clase TFormPartida

Crea el formulario que recoge la información que completa los datos del ejercicio 4: la pregunta que encabeza el ejercicio y los parámetros que rigen la estrategia de respuesta del computador.

Clase TFormJaque

Crea el formulario que recoge la información que completa los datos del ejercicio 5: la pregunta que encabeza el ejercicio y el número de movimientos de los que dispone el aprendiz para completarlo.

Clase TFormConfiguracion

Crea el formulario que recoge toda la información que necesita el Applet para funcionar y crea la página HTML en la que va metida el Applet.

10. Mejoras en el proyecto

En este apartado se discutirán distintos aspectos que podían haber sido tratados de manera distinta a como se han abordado produciendo mejoras en el desarrollo final del software.

Estas posibles mejoras se presentarán desde dos enfoques: mejoras en cuanto al diseño y mejoras en la implementación. En ambos casos se referirán al código generado para crear los ejercicios, ya que la implementación de la herramienta generadora es menos flexible en estos aspectos.

10.1 Mejoras en diseño

Analizando el diseño de clases de la aplicación, se observan las siguientes modificaciones posibles:

- La clase *casilla* podría haber sido sustituida por alguna estructura de datos más sencilla. Desde el punto de vista de uso de recursos es bastante ineficiente mantener 64 objetos casilla de los cuales puede ser que muchos estén vacíos. Podría haber sido la propia clase pieza la que gestionase las posiciones dentro del tablero, teniendo en cuenta solamente aquellas que están ocupadas. Evidentemente, esto conllevaría un replanteamiento total de la estructura de la aplicación que podría haber degradado el rendimiento en otros aspectos.
- Las clases que implementan los paneles contenidos en el panel principal podrían haberse fundido en una sola sin grandes esfuerzos. Así, la clase *PanelRespuesta* y la clase *PanelSolución* tienen comportamientos muy similares que podrían haberse controlado con el uso de parámetros. Por otra parte, tener una clase de panel para cada tipo de ejercicio proporciona mayor flexibilidad ante posibles cambios que afecten a un solo tipo de ejercicio.
- La clase tablero es la que mayor peso funcional tiene. Podría haber sido ventajoso haberla descompuesto en varias clases aumentando la claridad de código, la distribución de tareas y la modularidad.
- Los aspectos de encapsulación y visibilidad no se han tenido demasiado en cuenta durante el diseño de las clases, dejando así al descubierto algunos atributos y métodos que, por su

carácter, deberían haber pertenecido a un ámbito estrictamente privado.

- Aplicando los conocimientos adquiridos en asignaturas de patrones de diseño podrían haberse creado clases para implementar algunos que hubieran sido de gran utilidad: por ejemplo, una clase *Singleton* para las figuras de las piezas habría rebajado de 32 a 12 el número de imágenes necesarias; también la creación de clases *Iterator* habría hecho más eficiente el recorrido de las estructuras que guardan las piezas y las casillas.

Éstas y otras deficiencias que presenta el código se deben en general a dos razones fundamentales.

La primera es el hecho de haber tres personas trabajando por igual y simultáneamente sobre el mismo código, lo que ha originado bastantes conflictos en cuanto a los accesos a clases y a la representación de la información. Así vemos como, por ejemplo en la clase casilla tenemos atributos para representar lo mismo pero en dos formatos distintos, ya que en algunas clases se emplean con uno y en otras clases con otro, lo que podría haberse solucionado con simples métodos que transformasen los datos requeridos de una representación a otra. Como éste hay, lamentablemente, bastantes ejemplos de falta de coherencia en el código, que se podrían haber solucionado con una buena comunicación y mejor organización entre los miembros del equipo de desarrollo.

La segunda se deriva de la necesidad de adaptar código que encontramos en aplicaciones ya hechas al generado por nosotros mismos, lo que obligó a realizar algunos 'arreglos' que han perjudicado la legibilidad y la coherencia del código global. Tal vez haber dedicado mayor tiempo al estudio de las herramientas encontradas hubiera paliado estos efectos sobre el sistema, o quizás hubiera sido mejor – esta es nuestra opinión – haber generado todo el código nosotros mismos.

10.2 Mejoras en implementación

Centrándose en los aspectos más algorítmicos del sistema, las mejoras que más hubieran favorecido son las referentes a la función de estimación con la que se valora la calidad de las jugadas que elige el computador en sus respuestas.

Para dotar a esta función de más potencia la principal medida a adoptar sería la de valorar más características en ella, como controlar las diagonales libres, las columnas libres (aspectos positivos para el color q consiga dominar bien una columna o una diagonal libre, ya que da mayor movilidad a tus piezas y mayor capacidad de ataque), peón doblado (negativo para el color que tenga un peón doblado, porque resta movilidad), y ya en lo referente al Minimax, que se juegue para un nivel de mayor profundidad. Esto último se ha intentado conseguir, pero no ha habido mucho éxito. La causa posible haya sido una deficiente poda en nuestro árbol, haciendo esto que haya demasiadas ramas en el árbol con lo cuál muchos tableros que estimar, con lo cual no sea factible.

11. Evaluación del software y plan de pruebas

11.1 Plan de pruebas

1. Introducción y resumen de elementos y características a probar.

Los elementos software que se probarán con este conjunto de pruebas, son todos aquellos en los que el usuario introduce datos o interactúa con el sistema. Principalmente se trata de interfaces en los que el usuario ha de rellenar campos. Se comprobará que independientemente de lo que introduzca el usuario (incluido nada) la aplicación sigue comportándose correctamente, dando mensajes de error en los lugares adecuados.

2. Características que se van a probar:

Las características que se probarán son aquellas relacionadas con el caso de uso de Generar Ejercicio, que como se especificó anteriormente comprende varios subcasos de uso. Además, se van a probar elementos relacionados con el movimiento de las piezas que participan en el resto de los casos de uso (aparecen en los cinco ejercicios), al igual que ocurre con la prueba que se hará para comprobar que los mensajes de éxito o fracaso aparecen adecuadamente por pantalla. Los elementos software implicados son:

- Se comprobará que los datos considerados como obligatorios siempre se rellenan (obligando al usuario a hacerlo).
- Que en caso de que haya varias opciones disponibles, será obligatorio seleccionar alguna para poder continuar con la ejecución normal del programa.
- Que al introducir textos largos en las cajas de dialogo que aparecen cuando se pide qué mensajes se quieren mostrar en el ejercicio, no se producen fallos en la ejecución. Además, a pesar de la longitud de los textos, éstos se tienen que seguir visualizando sin problemas.
- Cuando el usuario mueve incorrectamente una pieza, el tablero no varía.

3. Enfoque general de la prueba:

Se realizarán pruebas de Sistema de Caja Negra basadas en los requisitos del sistema, con un análisis de valores límite (AVL).

4. Criterios de aceptación / fallo para cada elemento:

- Interfaces relacionados con el usuario: en los casos en los que

se indiquen que es obligatorio rellenar un campo, se aceptará el elemento si al intentar seguir la ejecución sin haberlo rellenado, el sistema informa de que no se han rellenado los campos. En otros casos se aceptará el elemento si no se permite seguir la ejecución hasta que se elija una opción.

- En el caso de que se haya rellenado un campo con un texto largo, este elemento se aceptará si se permite seguir normalmente con la ejecución, y a la hora de mostrar el ejercicio con los mensajes, estos se visualizan correctamente.
- En el caso del movimiento de una pieza, se aceptará este elemento si cuando intentamos mover la pieza a un lugar incorrecto el tablero no varía.

5. Criterios de suspensión y requisitos de reanudación: No se puede suspender.

6. Actividades de preparación y ejecución de pruebas:

   Antes de realizar estas pruebas, el usuario debe realizar las siguientes actividades:

   - Tener instalado correctamente la herramienta.
   - Tener instalado un navegador.

7. Necesidades de entorno:

   No existen necesidades de entorno. Esquemas de tiempo:

   El tiempo dedicado tanto al diseño como a la ejecución de las pruebas será de un par de horas.

8. Necesidades de personal y formación:

   Se utilizará una persona para llevar a cabo este plan de pruebas. Como las pruebas son de Caja Negra basadas en los requisitos, no hace falta que el usuario posea ninguna formación específica.

11.2 Especificación de diseño de pruebas

Identificación de cada una de las pruebas

- Caso de Prueba no seleccionar correctamente una subcarpeta de clases como directorio de salida. (ED-1)

Casos de prueba:

- Elegir una subcarpeta que no sea del directorio clases.
- Pulsar en el botón de siguiente.

Procedimiento a seguir: se selecciona la subcarpeta de salida que no es subcarpeta del directorio clases y se pulsa siguiente.

- Caso de Prueba no seleccionar ningún ejercicio en la pantalla de inicio(ED-2)

Casos de prueba:

- No seleccionar ninguna opción.
- Pulsar en el botón de siguiente.

Procedimiento a seguir: una vez seleccionada la carpeta, se pide que tipo de ejercicio se quiere generar. Para comprobar este elemento de prueba basta con pulsar en siguiente sin elegir ninguna opción.

- Caso de Prueba intentar guardar configuración sin piezas en el tablero(ED-3)

Casos de prueba:

- Se muestra el tablero vacío en alguno de los ejercicios.
- Se pulsa en guardar
- Se escribe un nombre de fichero
- Se pulsa en siguiente.

Procedimiento a seguir: consiste en guardar la configuración del tablero con ninguna pieza. Se ha de seguir los pasos indicados anteriormente. Para llegar al tablero inicial se puede seleccionar cualquier tipo de ejercicio.

- Caso de Prueba de no rellenar los campos que se mostrarán en ejercicio(ED-4)

Casos de prueba:

- Guardar correctamente la configuración del tablero siguiendo los pasos correctos.
- Cuando aparezca el formulario en el que se nos pide que rellenemos los campos, dejar al menos uno vacío.
- Pulsar en siguiente.

Procedimiento a seguir: guardar una configuración no vacía del tablero con un nombre. En ese momento aparece un formulario en

el que tendremos que intentar continuar habiendo dejado por lo menos un campo en blanco.

- Caso de Prueba de rellenar los campos que se mostrarán en ejercicio con cadenas largas (ED-5)

Casos de prueba:

- Introducir una cadena larga en alguno de los campos que pide el programa tras haber guardado una configuración.
- Pulsar en siguiente.

Procedimiento a seguir: se introduce una cadena larga en alguno de los campos (el tamaño de los mismos está limitado). A continuación se le da a seguir.

- Caso de Prueba mover pieza en ejercicio a posición incorrecta (ED-6)

Casos de prueba:

- Seleccionar una pieza del tablero (ejercicio generado).
- Colocar la pieza en lugar incorrecto.

Procedimiento a seguir: una vez se ha generado el ejercicio, se trata de mover una pieza del tablero a un lugar incorrecto, ya sea dentro del tablero (movimiento ilegal según reglas de ajedrez) o fuera del tablero.

11.3 Especificación de los casos de prueba

Caso de Prueba de no seleccionar correctamente una subcarpeta de clases como directorio de salida (E-1)

Elementos software a probar y características que se probarán:

Se probarán la interfaz relacionada con el usuario

La característica que se probará es que la carpeta seleccionada como directorio de salida ha de ser obligatoriamente una subcarpeta del directorio clases.

Especificación de las entradas requeridas y salidas esperadas:

entrada: seleccionar una carpeta que no sea subcarpeta del directorio clases. Pulsar siguiente.

salida: se muestra un error indicando al usuario que la carpeta ha de ser una subcarpeta del directorio clases.

Necesidades de entorno:

El ordenador ha de tener la herramienta instalada con los directorios apropiados.

Requisitos especiales de procedimiento: ninguno.

Dependencias entre casos de prueba: ninguna.

Caso de Prueba no seleccionar ningún ejercicio en la pantalla de inicio (E-2)

Elementos software a probar y características que se probarán:

Se probarán las interfaces relacionadas con el usuario, que permiten seleccionar un ejercicio ha generar.

La característica que se probará es que la interfaz no permite continuar con la ejecución del programa si no se elige algún tipo de ejercicio.

Especificación de las entradas requeridas y salidas esperadas:

entrada: pinchar en siguiente.

salida: ninguna

Necesidades de entorno:

El ordenador ha de tener la herramienta instalada con los directorios apropiados.

Requisitos especiales de procedimiento: ninguno.

Dependencias entre casos de prueba: depende del anterior caso de prueba (E-1)

Caso de Prueba intentar guardar configuración sin piezas en el tablero (E-3)

Elementos software a probar y características que se probarán:

Se probará la interfaz relacionada con el usuario.

La característica que se probará es que al intentar guardar una configuración del tablero sin ninguna pieza, el programa no nos permite continuar.

Especificación de las entradas requeridas y salidas esperadas:

entrada: pulsar en guardar con el tablero vacío.

salida: aparece un formulario que nos pide que introduzcamos el nombre del fichero.

entrada: introducir un nombre de fichero y pulsar guardar

salida: ningún cambio. No se puede pulsar en botón seguir

Necesidades de entorno:

El ordenador ha de tener la herramienta instalada con los directorios apropiados.

Requisitos especiales de procedimiento: ninguno.

Dependencias entre casos de prueba: ninguno.

Caso de Prueba de no rellenar los campos que se mostrarán en ejercicio (E-4)

Elementos software a probar y características que se probarán:

Se probará la interfaz relacionada con el usuario en el momento que pide que rellene los mensajes que se mostrarán al usuario en el ejercicio generado.

La característica que se probará es que la interfaz obliga a rellenar todos los campos, es decir, verifica que no haya ninguno vacío, y en el caso de que sí lo haya, se le informa al usuario que ha de rellenar todos los campos.

Especificación de las entradas requeridas y salidas esperadas:

entrada: introducir datos en las cajas de texto del formulario. Se ha de dejar alguno vacío.

salida: un mensaje de error que nos informa que se han de rellenar todos los campos.

Necesidades de entorno:

El ordenador ha de tener la herramienta instalada con los directorios apropiados.

Requisitos especiales de procedimiento: ninguno.

Dependencias entre casos de prueba: caso de prueba E-3

Caso de Prueba de rellenar los campos que se mostrarán en ejercicio con cadenas largas (E-5)

Elementos software a probar y características que se probarán:

Se probará la interfaz relacionada con el usuario, y además se comprobará que el ejercicio generado visualiza los mensajes correctamente

La característica que se probará es que los textos de los mensajes son guardados correctamente, y que éstos se muestran en el ejercicio de tal forma que se permite ver la totalidad de los mismos.

Especificación de las entradas requeridas y salidas esperadas:

entrada: introducir una cadena larga en alguno de los campos (por ejemplo, longitud máxima permitida para campos).

salida: a la hora de mostrar el ejercicio, los mensajes aparecen correctamente en el Applet.

Necesidades de entorno:

El ordenador ha de tener la herramienta instalada con los directorios apropiados y un navegador.

Requisitos especiales de procedimiento: ninguno.

Dependencias entre casos de prueba: emplea mismo formulario que caso de prueba E-4

Caso de Prueba mover pieza en ejercicio a posición incorrecta (E-6)
Elementos software a probar y características que se probarán:

Se probará la interfaz relacionada con el usuario en el ejercicio generado además de la lógica interna del Applet

La característica que se probará es la comprobación que se hace en el Applet cuando se mueve una pieza, no validando aquellos movimientos incorrectos.

Especificación de las entradas requeridas y salidas esperadas:

entrada: mover una pieza del tablero a una posición ilegal (fuera del tablero o no de acuerdo con las reglas de ajedrez)

salida: el movimiento no se realiza. El jugador sigue teniendo el control del programa, que espera a que realice movimiento válido.

Necesidades de entorno:

El ordenador ha de tener la herramienta instalada con los directorios apropiados y un navegador.

Requisitos especiales de procedimiento: ninguno.

Dependencias entre casos de prueba: ninguno

11.4 Históricos de las pruebas

Histórico de ejecución de Caso de Prueba no seleccionar correctamente una subcarpeta de clases como directorio de salida

Descripción de la prueba

Se probarán las interfaces de usuario relacionadas con la selección de una capeta de salida. La ejecución de la prueba es guiada por el documento de pruebas ED-2.

Se selecciona una carpeta que no es subcarpeta de clases y se le da a continuar. Resultado: se le informa al usuario por medio de un mensaje que la carpeta ha de ser subclase del directorio clases. Ejecución correcta.

Histórico de ejecución de Caso de Prueba no seleccionar correctamente una subcarpeta de clases como directorio de salida (H-1)

Descripción de la prueba

Se probará la interfaz de usuario relacionada con la selección de uno de los ejercicios que se quiere generar. La ejecución de la prueba es guiada por el documento de pruebas ED-2.

No seleccionar ninguna opción e intentar pulsar en siguiente. Resultado: la aplicación no hace nada y sigue a la espera de que se elija una opción. Ejecución correcta.

Histórico de ejecución Caso de Prueba intentar guardar configuración sin piezas en el tablero (H-3)

Descripción de la prueba

Se intentará guardar una configuración del tablero sin ninguna pieza, el programa no nos permite continuar. La ejecución de la prueba es guiada por el documento de pruebas ED-3.

Pulsar en guardar configuración sin haber puesto ninguna pieza en el tablero. Escribir un nombre de archivo cuando el sistema nos muestre el formulario. Resultado: el sistema no nos deja dar al botón de siguiente, exigiéndonos una configuración del tablero no vacía. Ejecución correcta.

Histórico de Caso de Prueba de no rellenar los campos que se mostrarán en ejercicio (H-4)

Descripción de la prueba

Se probará la interfaz relacionada con el usuario en el momento que pide que rellene los mensajes que se mostrarán al usuario en el ejercicio generado.

La ejecución de la prueba es guiada por el documento de pruebas ED-4.

Dejar vacío alguno de los campos que se nos pide rellenar para mostrar en el ejercicio. Resultado: Mensaje que nos indica que es necesario rellenar todos los campos. Ejecución correcta.

Histórico de ejecución de Caso de Prueba rellenar los campos que se mostrarán en ejercicio con cadenas largas (H-5)

Descripción de la prueba

Se probará la interfaz relacionada con el usuario, y además se comprobará que ejercicio generado visualiza los mensajes correctamente. La ejecución de la prueba es guiada por el documento de pruebas ED-5.

Introducir cadenas largas en el formulario de datos en el que se nos pide rellenar los mensajes que se mostrarán al usuario. Resultado: el sistema genera los ejercicios mostrando los mensajes correctamente en el Applet, adaptando los tamaños de los campos destinados a tal fin. Ejecución correcta.

Histórico de ejecución Caso de Prueba mover pieza en ejercicio a posición incorrecta (H-6)

Descripción de la prueba

Se probará la interfaz relacionada con el usuario en el ejercicio generado además de la lógica interna del Applet. La ejecución de la prueba es guiada por el documento de pruebas ED-6.

Se intenta mover repetidamente piezas a posiciones incorrectas. También se intenta mover piezas del contrario. Resultado: el sistema devuelve las piezas a su posición original sin cambiar el turno del jugador. Ejecución correcta.

12. Glosario

Applet	Un Applet es una pequeña aplicación software, normalmente en un lenguaje de programación Java.
Ejercicio	Ejercicio de Ajedrez para ser resuelto. Se genera con herramienta generadora y se resuelve en un Applet.
Herramienta generadora	Aplicación en C++ que sirve para generar ejercicios “tipo” de Ajedrez.
Minimax	Un algoritmo usado en juegos con adversario y con información perfecta, donde se supone que el adversario juega óptimamente.
Piolín	Personaje de dibujos animados que sirve para mostrar al alumno a dónde tiene que conseguir llegar moviendo la pieza.
Mina	Dibujo que se utiliza en el tablero de ajedrez para representar que no se puede pasar por esa casilla
Meta	Destino del ejercicio de mover pieza. La pieza tiene que llegar a este punto en el menor número de pasos.
Configuración	Se refiere a como están dispuestas las piezas en el ejercicio que se genera con la herramienta. También se refiere a los mensajes que se tienen que mostrar en el Applet.
Archivo Tab	Extensión empleada para guardar ficheros de configuración de los ejercicios.
Parámetros configuración	Archivos que se pasan al Applet para que pueda generar el ejercicio según se especificó con la herramienta generadora de ejercicios.
Componente	Paquete que integra un conjunto de funcionalidades
Formulario	Ventana en la que existen campos para ser rellenados por el usuario
Interfaz	Clase abstracta en la que sólo se definen los métodos que se definirán en otras clases

13. Otros materiales elaborados

Con objeto de mostrar de una manera más amena y práctica los resultados de APDrez, se ha decidido crear un sitio Web en donde se encontrarán multitud de ejemplos de uso. Esta página sirve además como ejemplo de cómo utilizar los ejercicios creados.

La dirección es: http://ajedrezsi.iespana.es/

14. Bibliografía

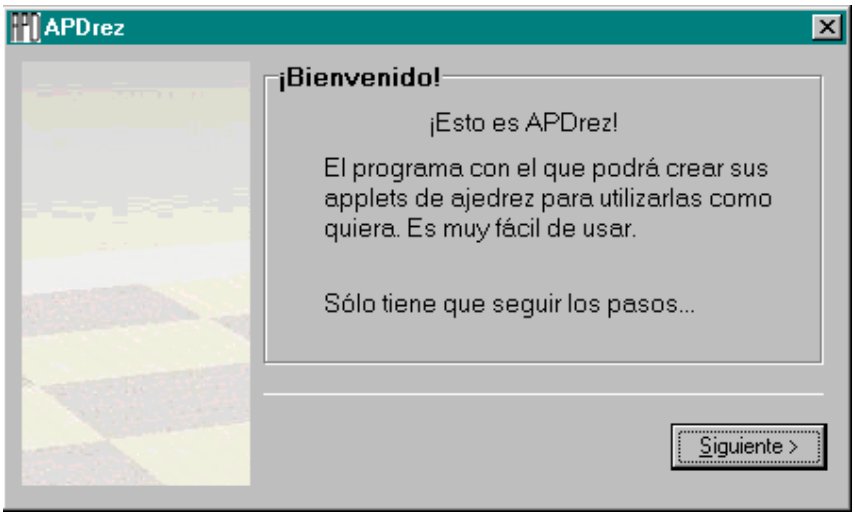
- Antonio Gude: *Escuela de Ajedrez.* Tutor 1988
- Pablo Aguilera: *Ajedrez para jóvenes.* Alianza Editorial 2001
- *Ajedrez infantil*. 2ª Edición. editorial paidotribo
- *Ajedrez con los Simpsons*. Planeta de Agostini
- ChessMaster 7000
- http://www.cjkware.com/chess/
- http://ajedrez.educared.net/
- http://www.enpassant.dk/chess/diaeng.htm
- http://www.mailchess.de/pgntojse.HTML
- http://www.mallardsoft.com/chessboard.HTML
- http://www.HTMLpoint.com/guida/HTML_16.htm
- http://www.enpassant.dk/chess/diaeng.htm
- http://www.very-best.de/pgn-spec.htm#1
- http://foro.Webexperto.com/viewforum.php?f=5
- www.geocities.com/zodiamoon/amyan/comofuncionan.HTML
- http://mailWeb.udlap.mx/~asanchez/ia/minimax3.HTML
- www.resplendence.com/chessbrd
- Roger S. Pressman: *Ingeniería del software. Un enfoque práctico*
- José A. Cerrada Somolinos: *Introducción a la ingeniería del software.* Editorial Centro de Estudios Ramón Areces, S.A 2000

15. Apéndice: Manual de usuario

Se presenta a continuación una breve guía de manejo de la herramienta. Para ello se generará un ejercicio por cada tipo de los posibles, explicando cada una de las ventanas de la aplicación, su razón de ser y las funcionalidades que incluye.

Generación de un ejercicio de tipo 1

La primera pantalla que se muestra al iniciar la aplicación tiene el siguiente aspecto:

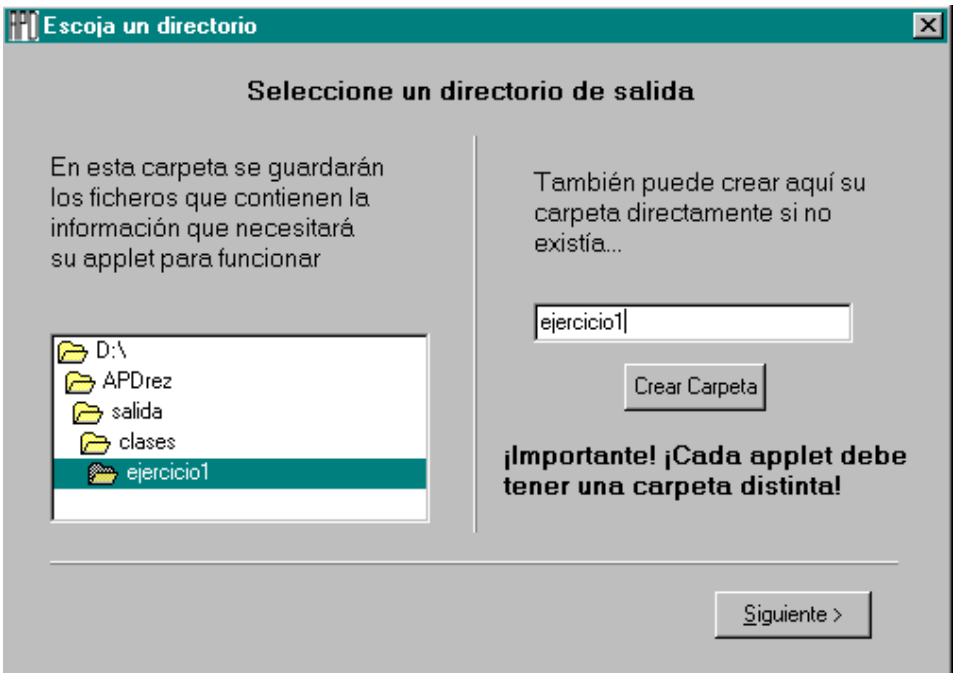


En ella se le da la bienvenida al usuario del programa haciendo una breve presentación de la herramienta.

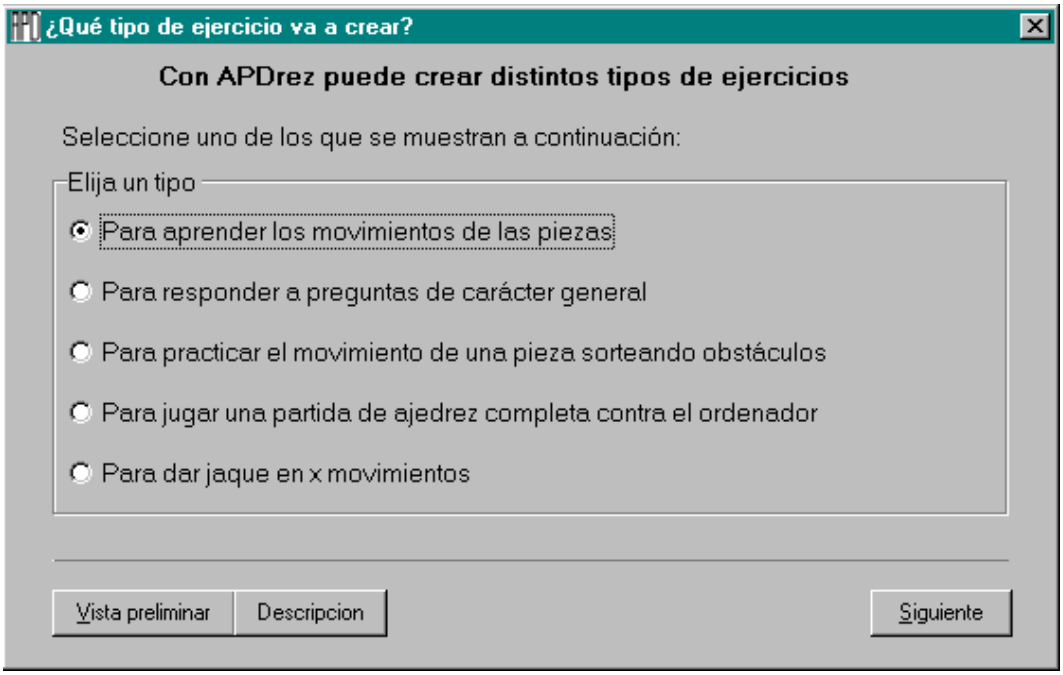
A continuación la herramienta le solicitará al usuario una carpeta en la que se guardarán los ficheros que contienen toda la información que necesitará el applet para funcionar correctamente. La herramienta le da la posibilidad de elegir una de las que haya dentro de la carpeta 'clases' o bien de crearla bajo este directorio, en cuyo caso pasará a ser seleccionada como la elegida en el momento de su creación. No se admitirán como válidas aquellas carpetas que no sean hijas directas de la carpeta clases.

Es importante que el usuario cree una carpeta distinta para cada applet que vaya a generar. De otro modo, se podrían sobrescribir los archivos que contienen la información de los ejercicios, lo que originaría un mal funcionamiento de los mismos.

Una vez elegida la carpeta correctamente, se activará el botón 'Siguiente' que conduce hacia la siguiente pantalla de la aplicación.

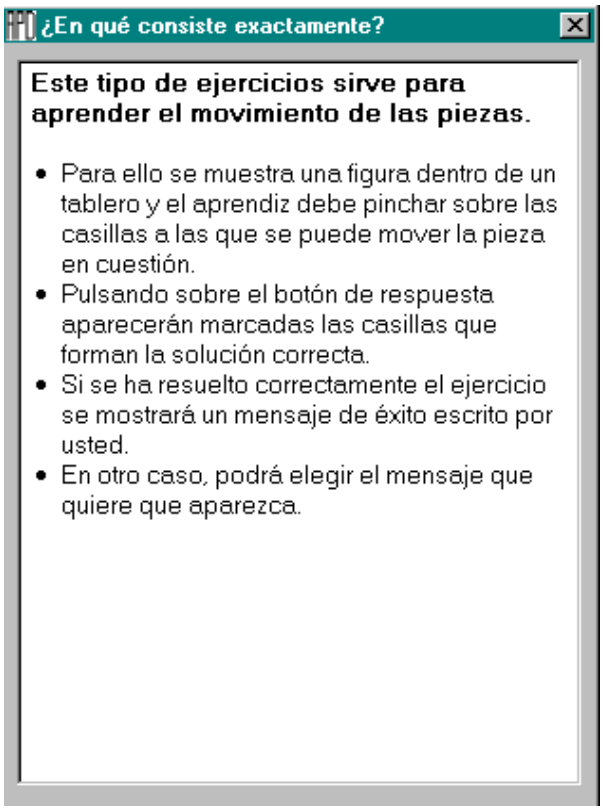


Tras crear una carpeta para el primer ejercicio se elegirá el tipo de applet que se va a crear. Se ofrecen hasta cinco posibilidades. Solamente podrá avanzar el usuario en la creación cuando haya seleccionado una de entre ellas.



Se decide crear un ejercicio del primer tipo.

Si el usuario quiere saber un poco más del funcionamiento del applet creado, pulsando sobre el botón 'Descripción' aparecerá una nueva ventana en donde se explica de qué trata el ejercicio, qué parámetros podrá configurar, y cómo actuará una vez que se haya creado.

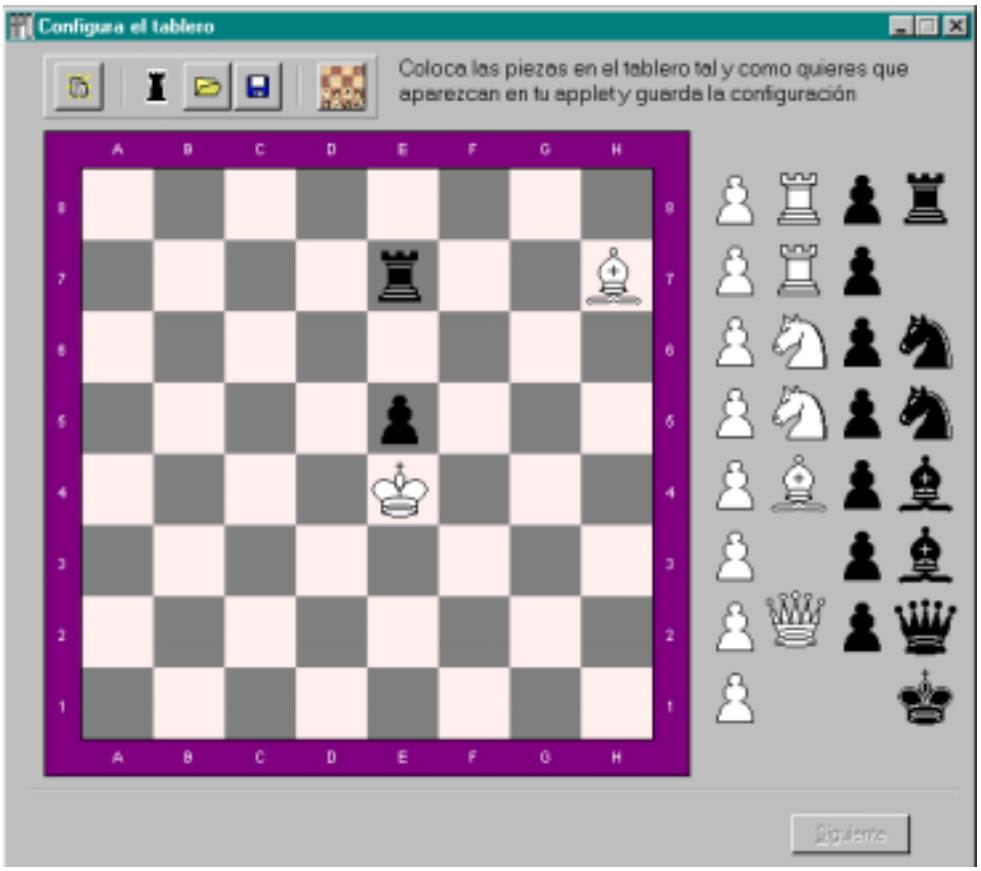


En la siguiente pantalla el usuario deberá poner las piezas en el tablero tal y como quiere que aparezcan en el ejercicio.

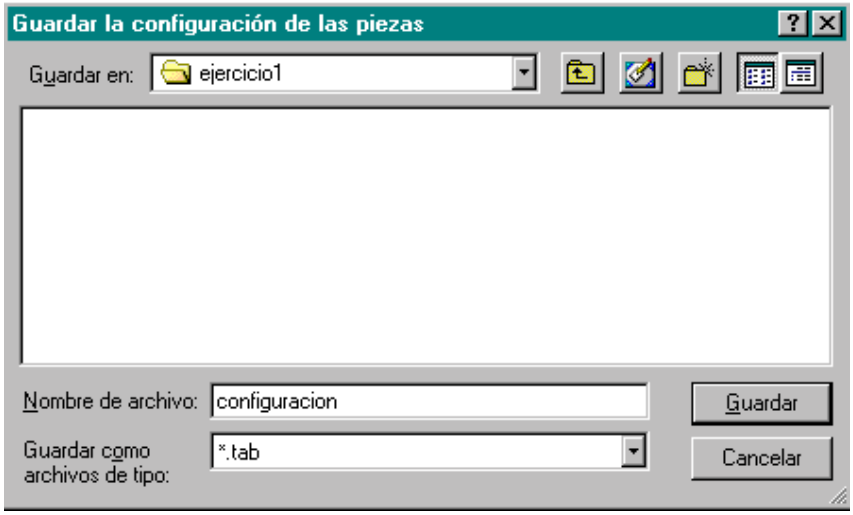
Una vez colocadas a su gusto, deberá guardar la configuración para poder avanzar a la siguiente pantalla. El formato de los archivos que contienen la configuración es uno creado a tal efecto al que se le ha asignado la extensión 'tab' y que es en realidad un fichero de texto que contiene tantas líneas como piezas hay sobre el tablero, y en cada línea identificadores para la pieza y la casilla que ocupa.

El usuario dispone además de otras posibilidades, como abrir una configuración guardada previamente – permitiendo solamente recuperar archivos formato 'tab' – o bien colocar todas las piezas en el tablero tal y como están al iniciar una partida de ajedrez o quitar todas las piezas que haya podido colocar previamente.

Solamente cuando el usuario haya guardado una configuración no vacía de las piezas, podrá acceder a la siguiente pantalla.



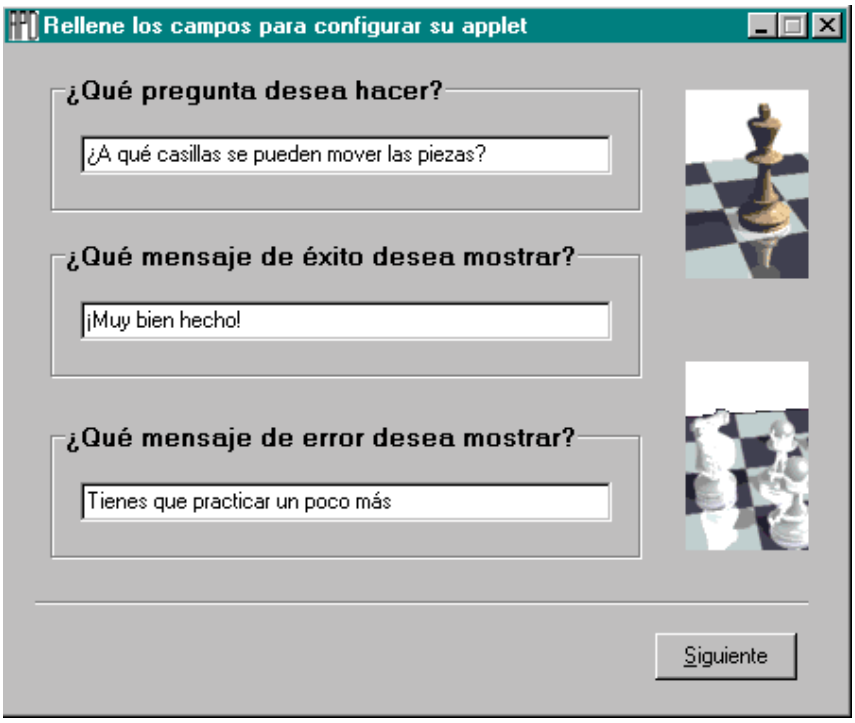
El usuario decide colocar las piezas como se muestran en la figura.



Y guardar la colocación en el fichero 'configuracion.tab'.

Cuando el usuario pulsa el botón siguiente, se creará en el directorio elegido al principio un archivo nuevo llamado 'casillas_solucion.txt' que contendrá todas las casillas a las que se pueden mover las piezas que hay sobre el tablero.

A continuación el usuario deberá rellenar otros campos necesarios para el ejercicio: la pregunta que lo encabezará, el mensaje que desea dar cuando el aprendiz lo resuelva correctamente y el que se mostrará en caso contrario.



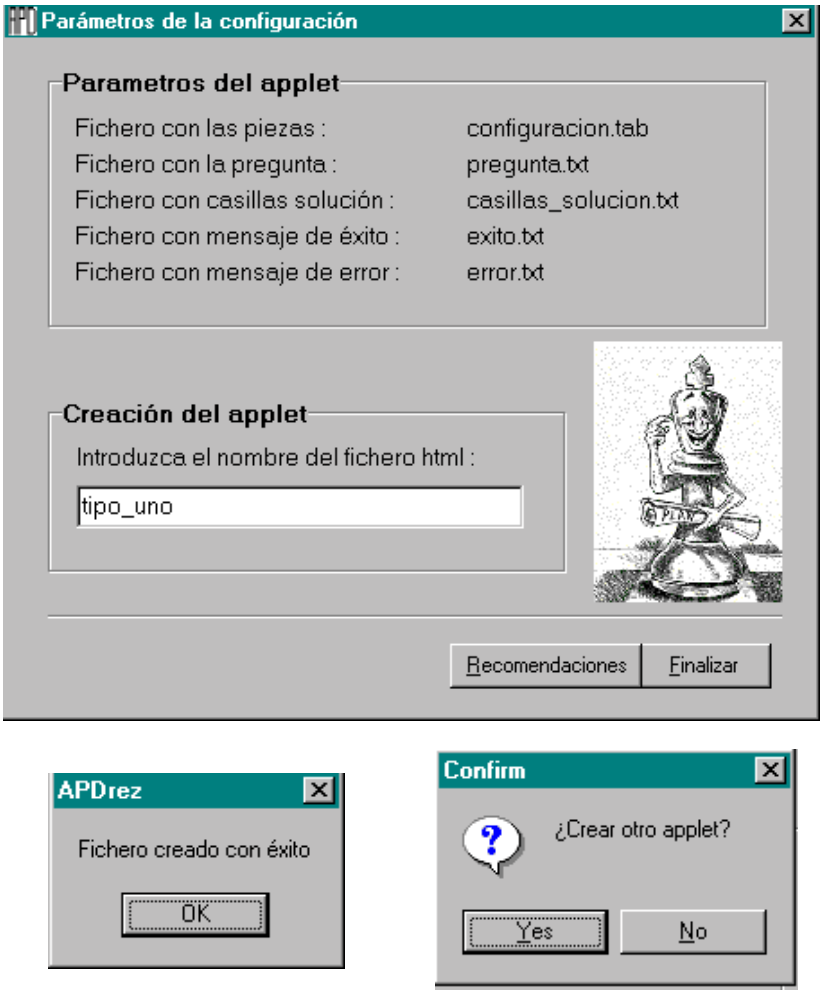
Solamente cuando se hayan rellenado estos campos correctamente el usuario podrá avanzar a la siguiente pantalla.

En ella se muestran los parámetros del ejercicio - nombre de los archivos que contienen la información - y por último se solicita el nombre de la página html que contendrá el applet final.

El usuario puede elegir leer las 'Recomendaciones', en cuyo caso se despliega una nueva ventana con algunos consejos para usar el applet.

Una vez elegido un nombre, la herramienta generará el código html necesario para la página contenedora, dando un mensaje de éxito si el fichero se ha creado sin problemas.

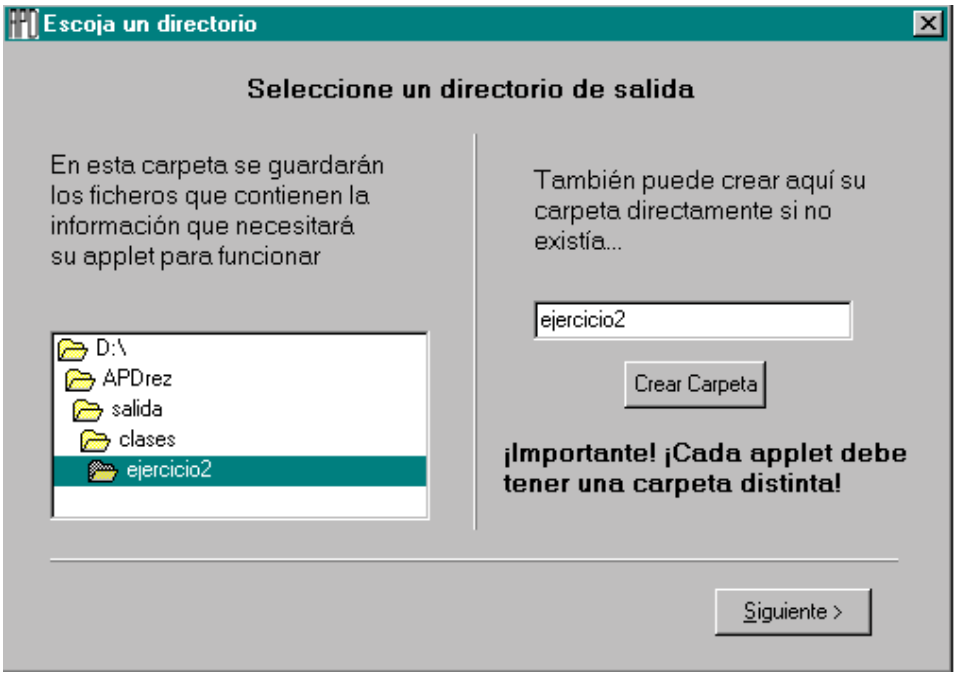
Por último el usuario puede elegir entre crear otro ejercicio o salir de la aplicación.



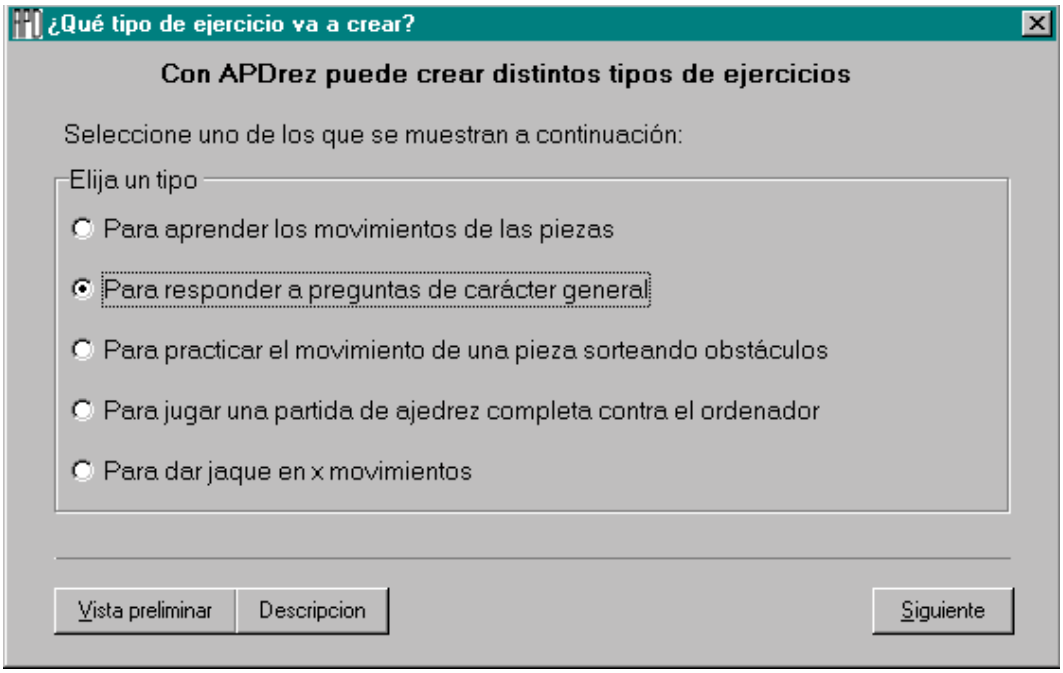
Y el resultado es el siguiente:

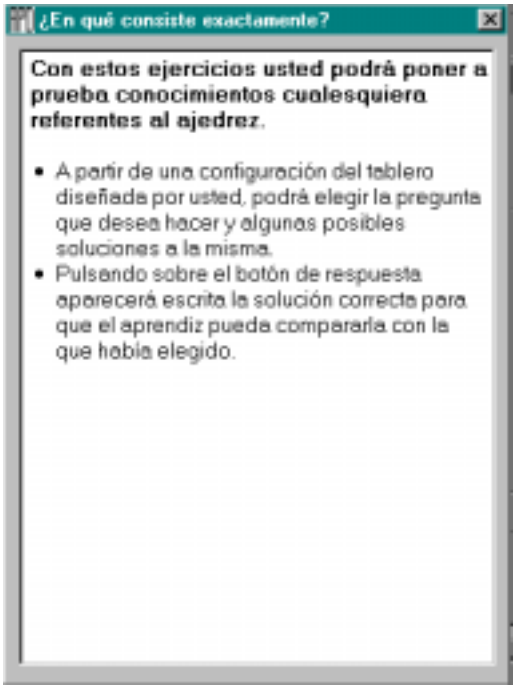
Generación de un ejercicio de tipo 2

Nuevamente el usuario deberá elegir una carpeta en la que guardar los ficheros que contienen la configuración del ejercicio.



Se deberá elegir esta vez la segunda de las cinco opciones posibles.



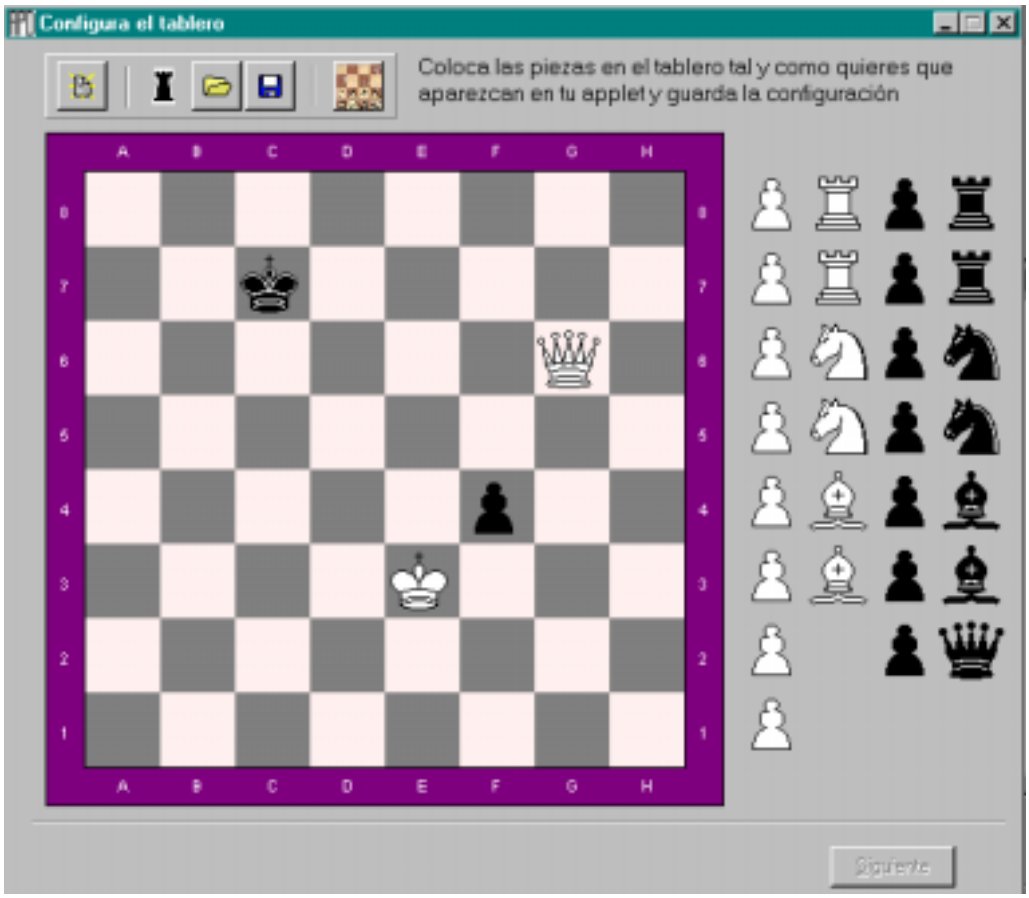


De la misma manera que antes, pulsando sobre el botón 'Descripción' el usuario obtiene un breve texto explicativo a cerca del ejercicio.

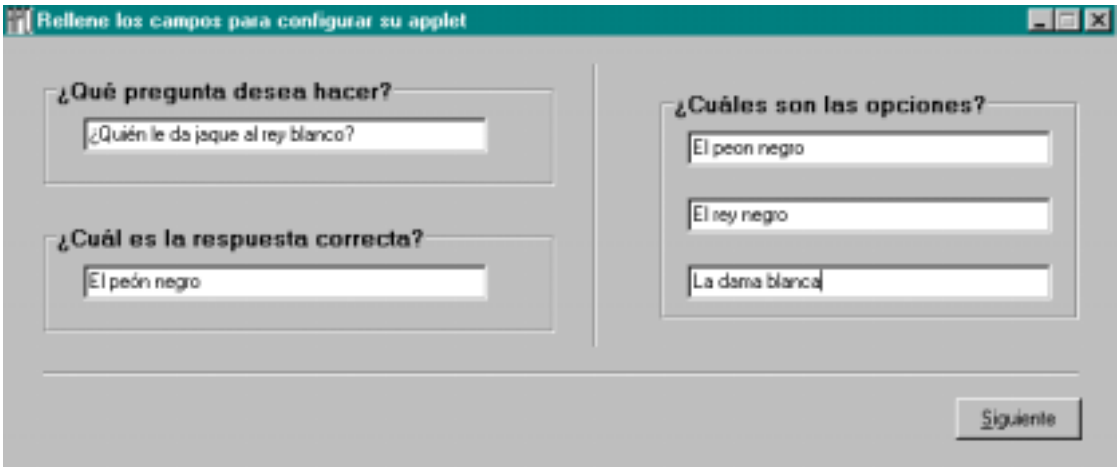
Una vez seleccionado el tipo, el usuario deberá escoger la configuración que aparecerá en el tablero del applet generado.

Guardada la colocación de las piezas, podrá continuar hacia la siguiente pantalla.

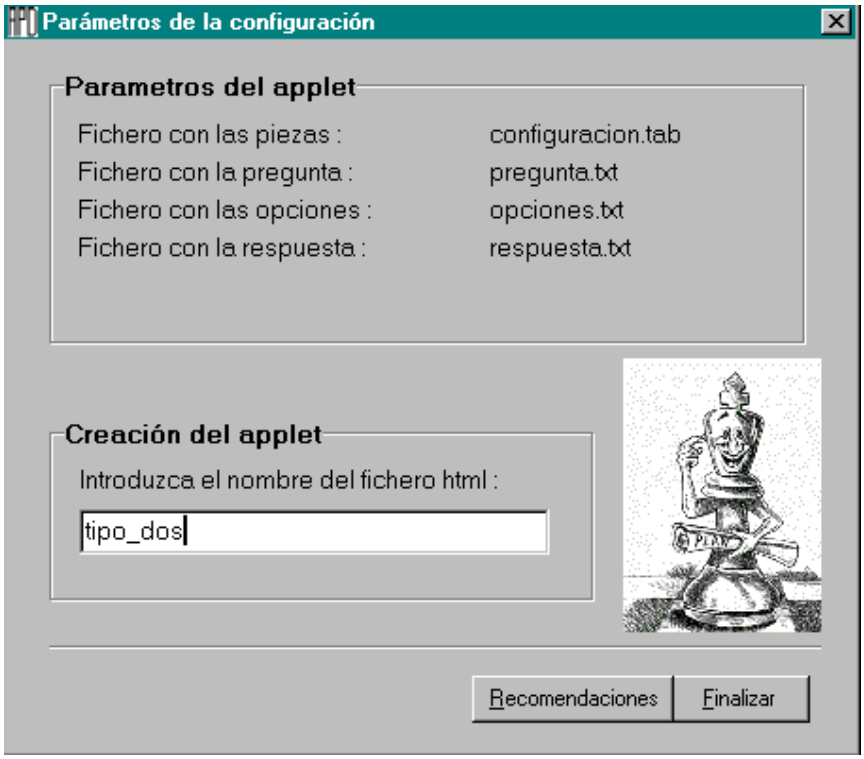
Se supone que el usuario ha puesto las piezas en el tablero como se muestra en la figura de abajo.



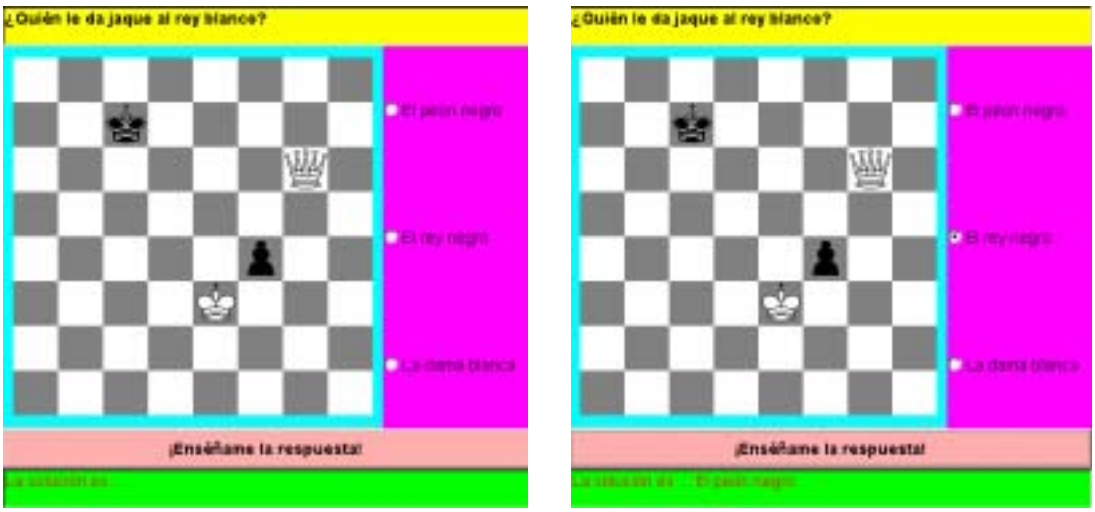
El usuario deberá rellenar los campos que se muestran en esta pantalla para completar la información del ejercicio. Deberá poner la pregunta que desea hacer – a partir de la configuración guardada previamente – las opciones que le dará al aprendiz para que elija – hasta 3 posibles – y por último la respuesta correcta a la pregunta.



Finalmente, en la ventana de configuración aparecen los ficheros que contienen la información del applet y se solicita un nombre para el archivo html que lo contendrá y que será creado al pulsar el botón 'Finalizar'.

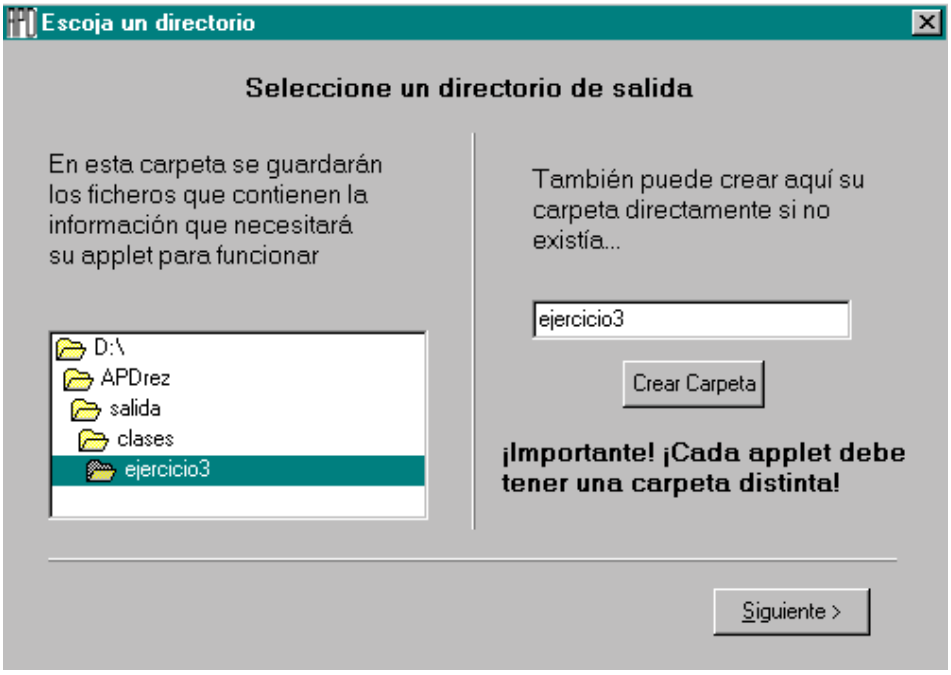


El resultado obtenido se muestra a continuación:

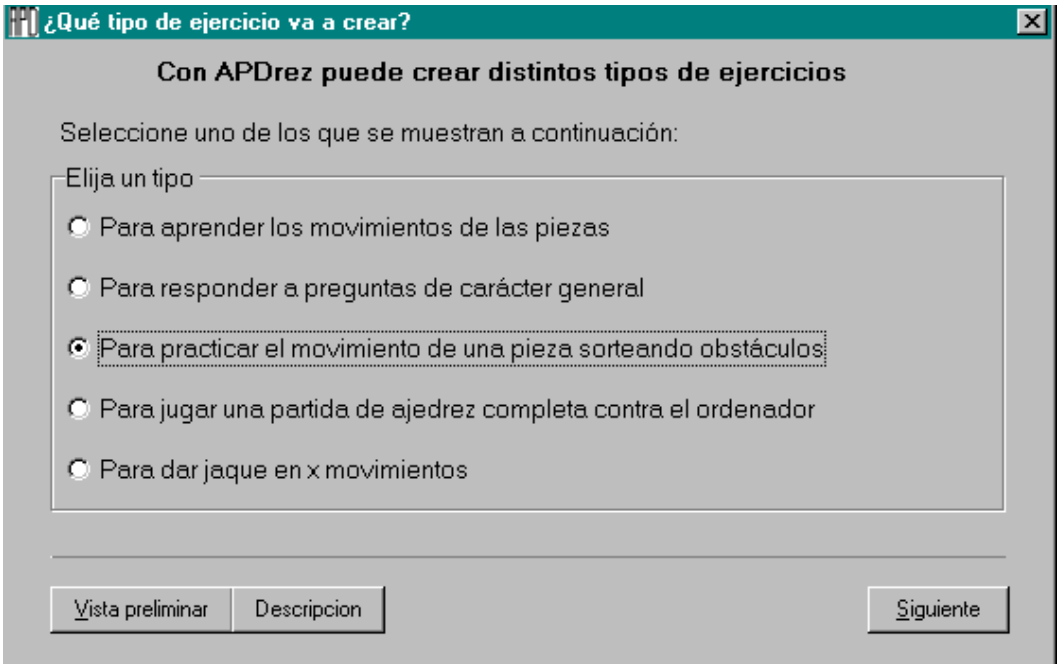


Generación de un ejercicio de tipo 3

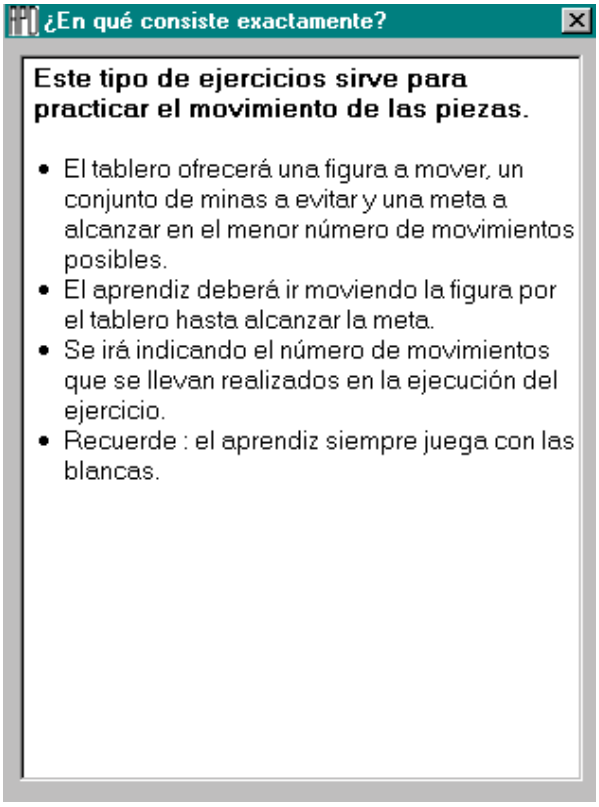
El usuario deberá crear nuevamente una carpeta para guardar los archivos que contienen información del applet.



Y elegir crear un ejercicio del tercer tipo.



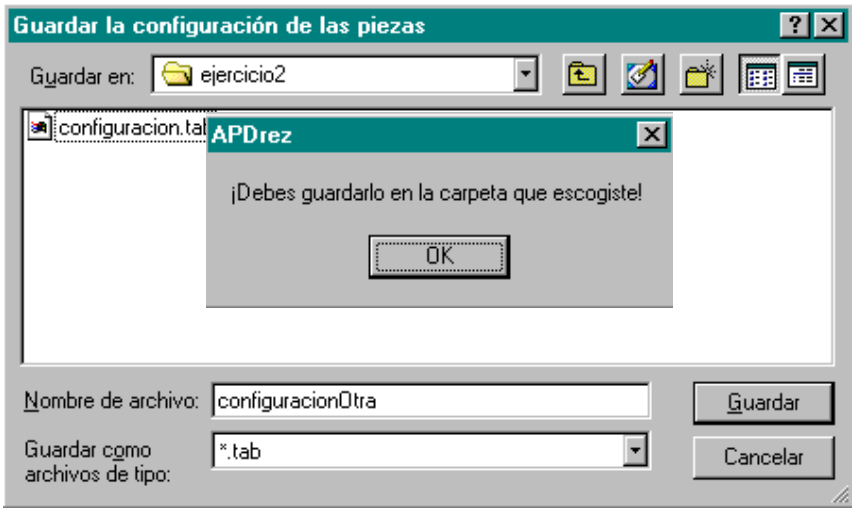
Tras haber leído la descripción del ejercicio y marcado la tercera casilla, el usuario colocará, no sólo las piezas dentro del tablero, si no también las minas que deberá sortear el aprendiz y la meta a alcanzar.



Cuando el usuario haya acabado y desee guardar la configuración se crearán tres archivos: uno con la colocación de las piezas, otro con la de las minas y un último con la de la meta.

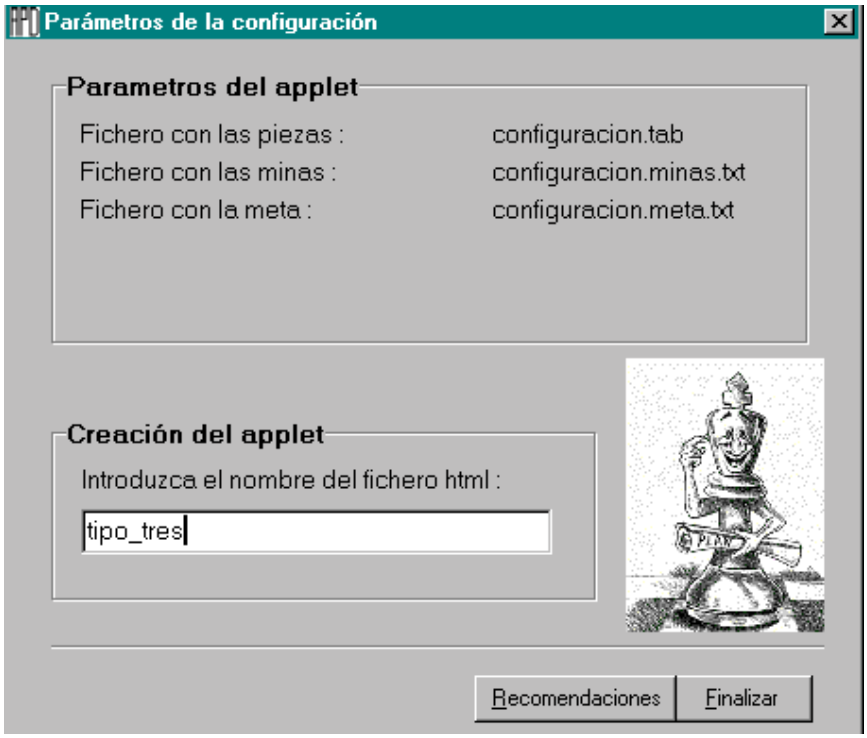
De la misma manera cuando intente recuperar una configuración en este tipo de ejercicios, se recuperarán además de la de las piezas – si es que existen – las de las minas y la meta.

Obsérvese que la herramienta no permite guardar las configuraciones en otro directorio que no sea el que el usuario haya elegido en el primer paso de la aplicación.

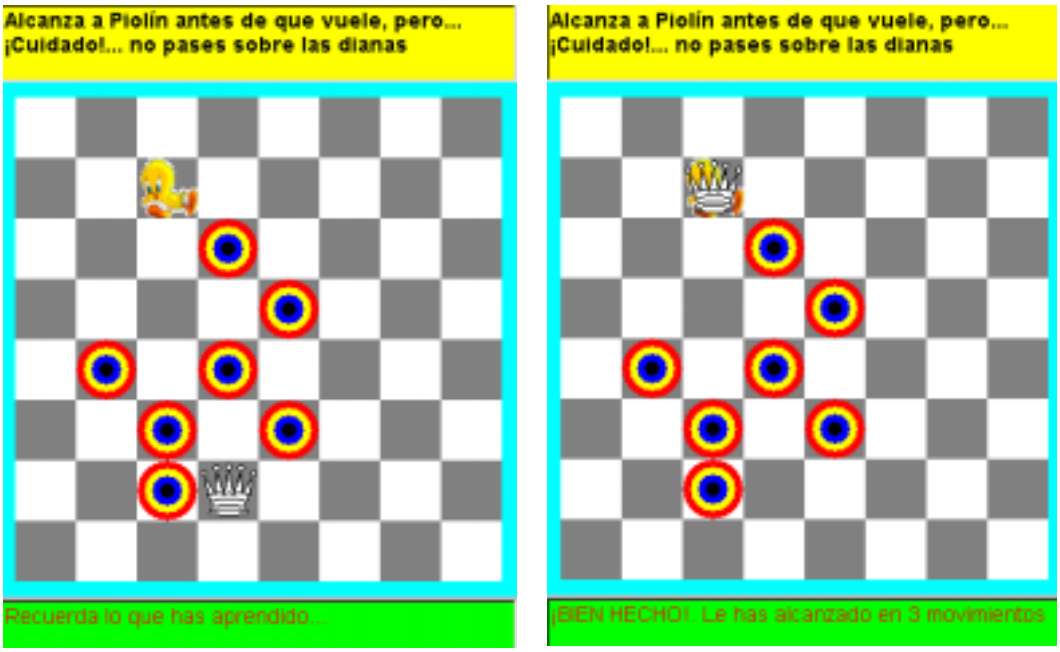


Cuando el usuario escoja la carpeta correcta, se crearán tres ficheros: configuracion.tab, configuracion.minas.txt y configuracion.meta.txt.

Finalmente, se muestran los parámetros de la configuración y se creará la página html final.

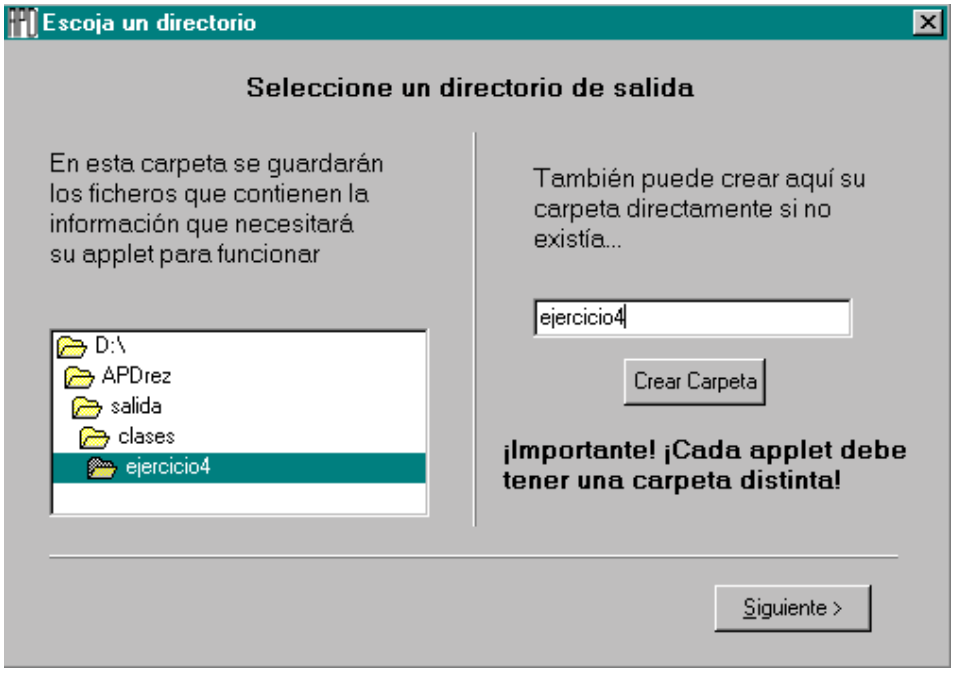


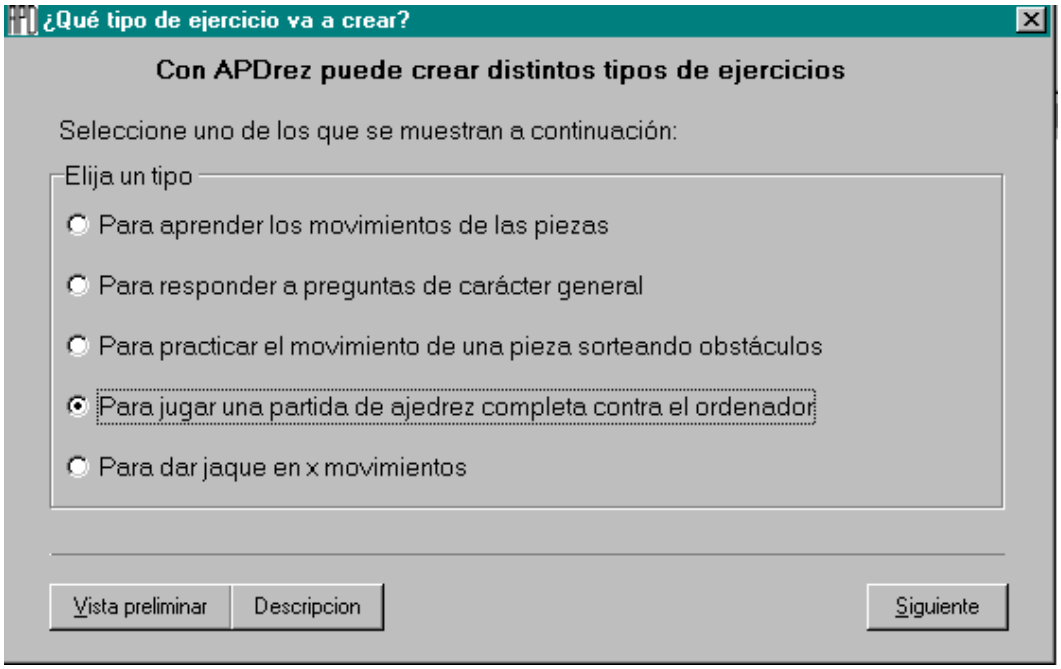
El resultado final es el siguiente:



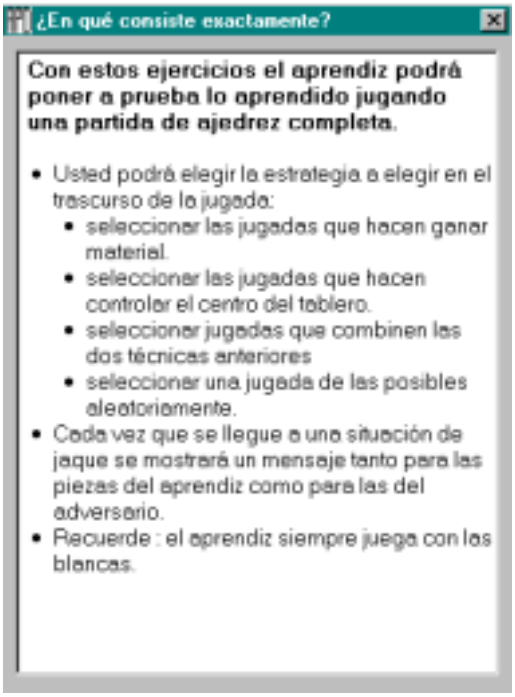
Generación de un ejercicio de tipo 4

Tras elegir el directorio de trabajo y seleccionar el cuarto como el tipo de ejercicio a crear, el usuario deberá colocar las piezas en el tablero para que el aprendiz pueda jugar una partida de ajedrez.





Si el usuario lee la descripción del ejercicio correctamente, verá que el aprendiz siempre juega con las blancas, lo que tendrá que tener en cuenta en la colocación de las piezas.



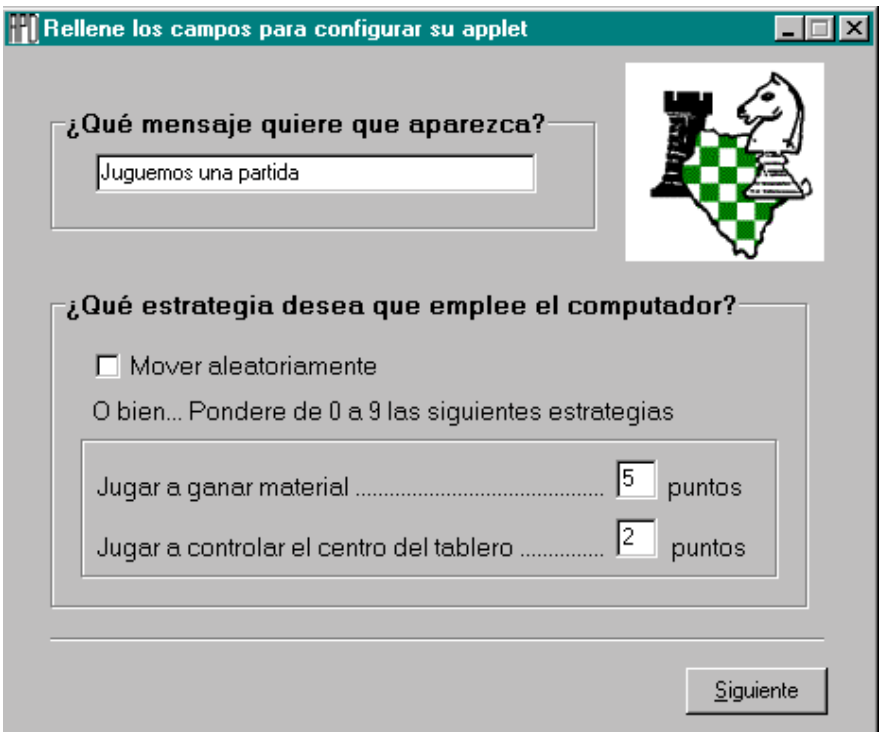
También puede optar por colocar todas las piezas en el sitio que le corresponden inicialmente con el botón derecho de la barra de herramientas.

Una vez guardada la configuración, se activará el botón que le permite al usuario seguir completando los parámetros de su ejercicio.

El usuario deberá escribir el mensaje que quiere que encabece la partida y la estrategia que seguirá el computador para responder a las jugadas del aprendiz.

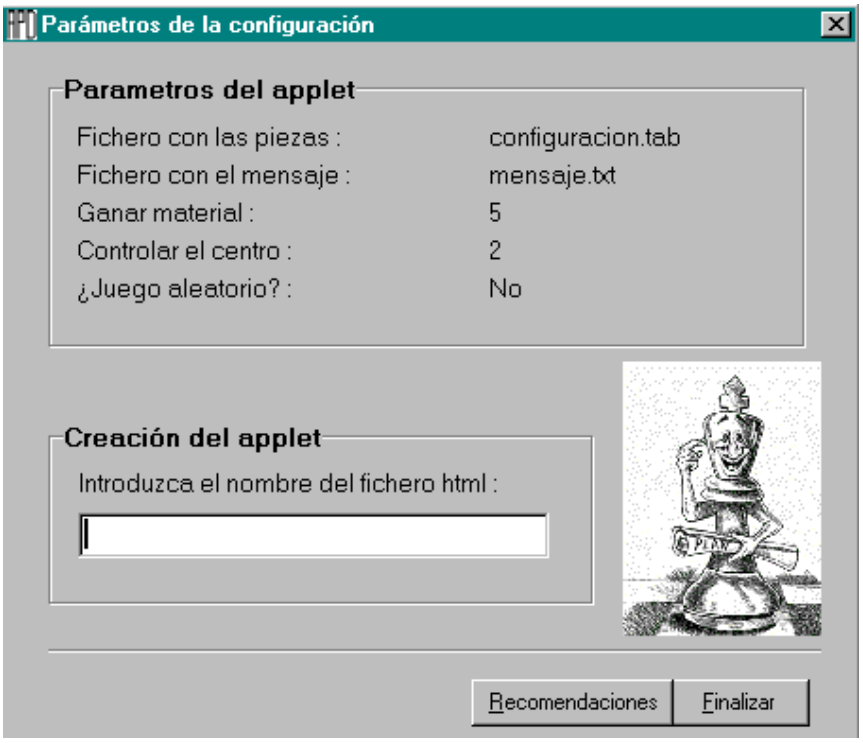
Si marca la casilla 'Mover aleatoriamente', el computador elegirá un movimiento cualquiera de entre todos los que le sean posibles hacer. Por el contrario, si esta casilla está desmarcada, podrá elegir puntuando de 0 a 9 a qué quiere que le dé más importancia: a ganar material, a controlar el centro del tablero o bien una combinación de ambas.

Los números que se muestran por defecto en las casillas de puntuación han sido calculados para que el computador valore las jugadas de forma que se escojan siempre las más propicias dentro de las posibles.

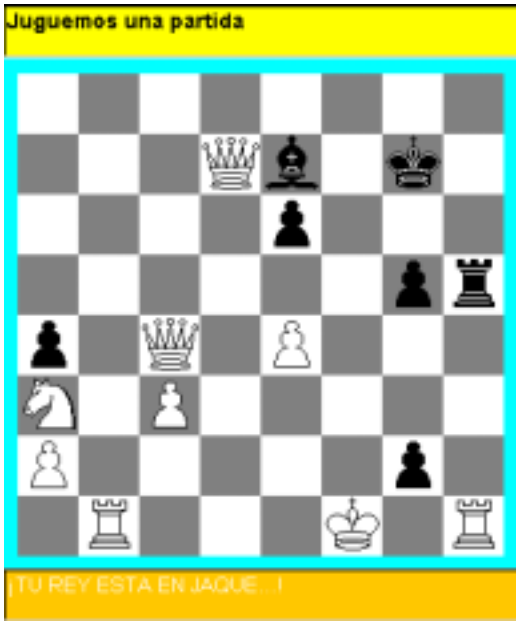


Las opciones de mover aleatoriamente y ponderar las estrategias para ganar material y / o controlar el centro del tablero son excluyentes. El usuario sólo podrá elegir una de las dos.

Una vez rellenados los campos correctamente, el usuario podrá leer los parámetros de configuración de su applet y elegir el nombre del fichero html que contendrá el ejercicio generado.

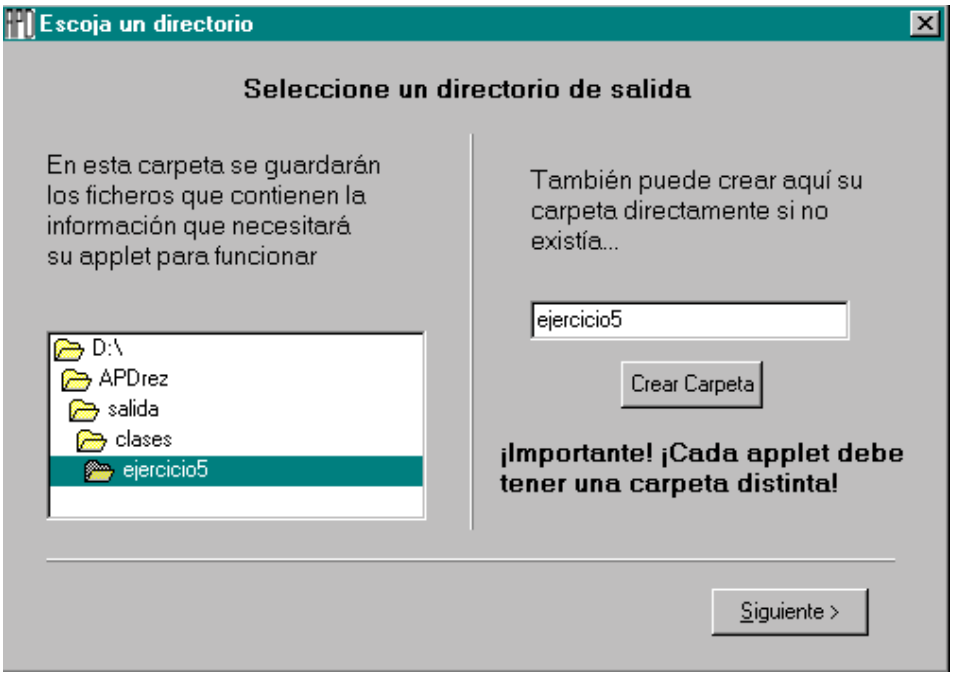


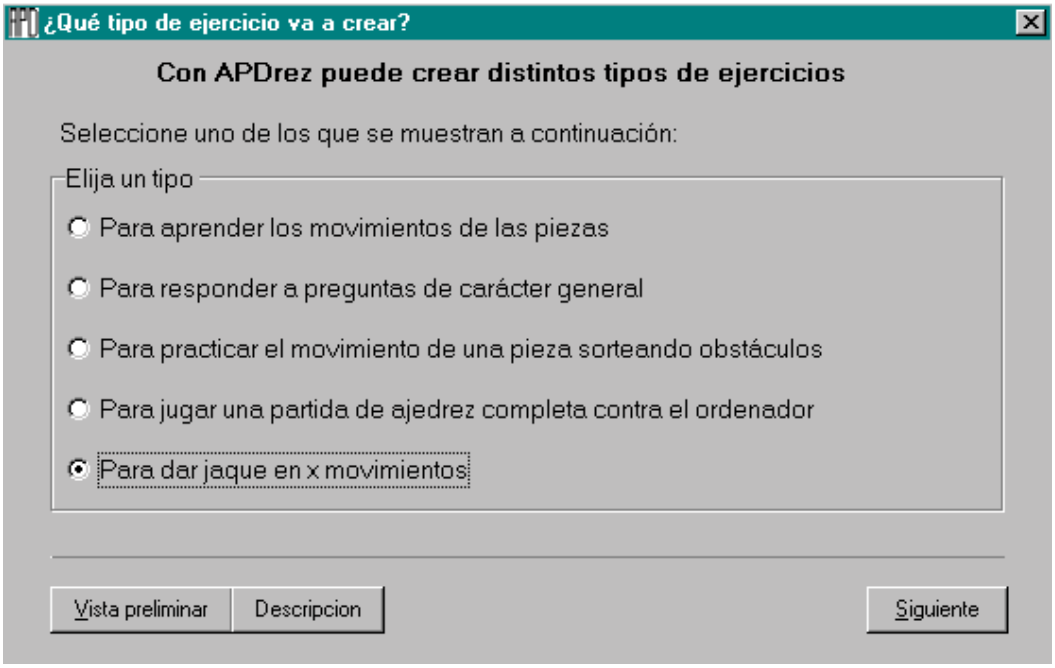
El ejercicio generado es el que se muestra a continuación:



Generación de un ejercicio de tipo 5

Como siempre, el usuario escogerá el directorio de trabajo y el tipo de ejercicio que va a crear – el quinto en este caso – .

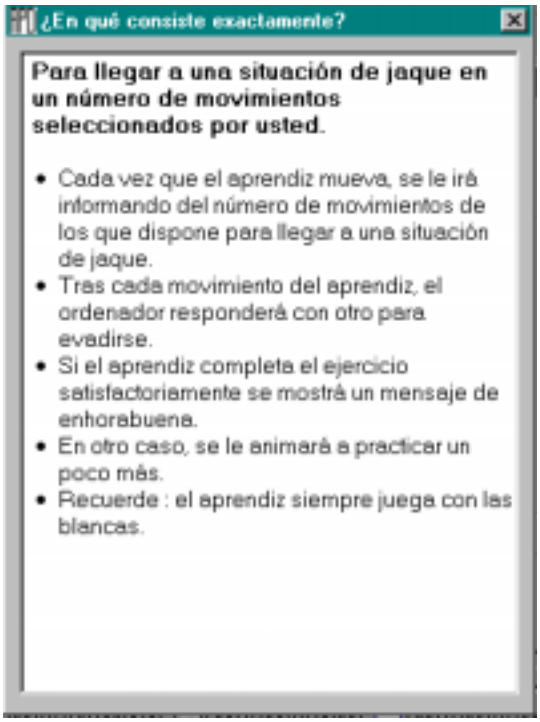


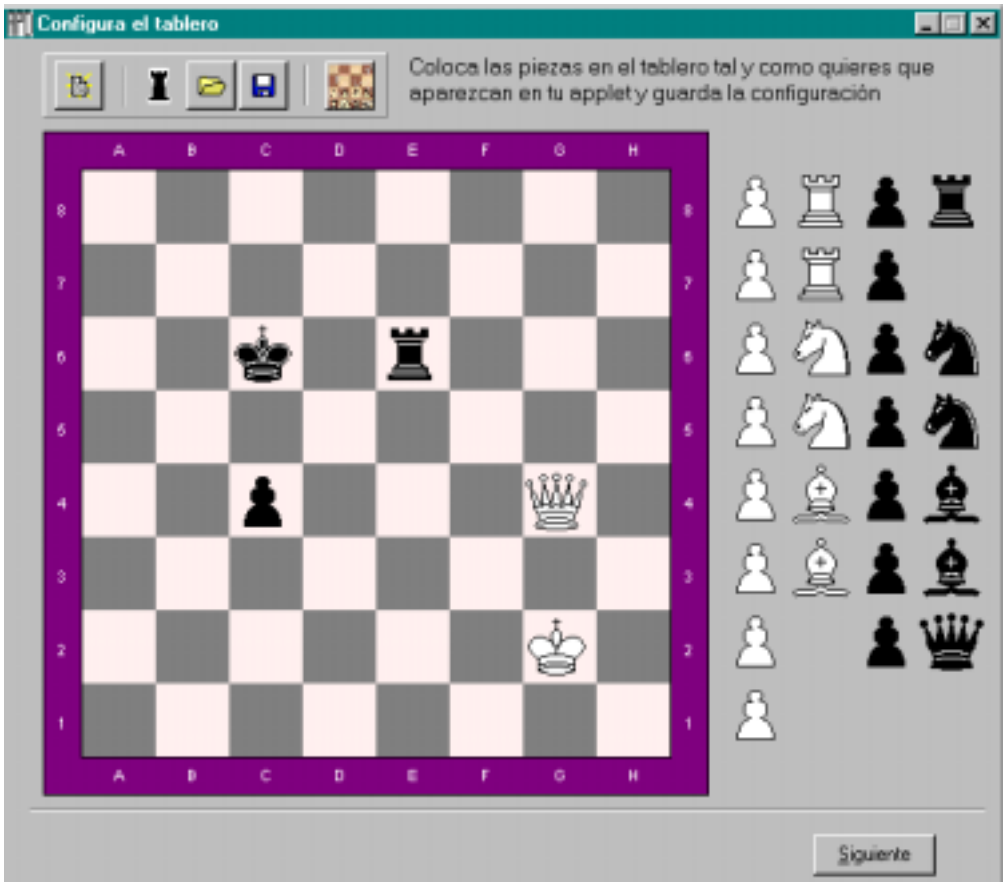


El usuario deberá colocar las piezas en el tablero de ajedrez en una configuración tal que sea posible darle jaque al rey negro – el aprendiz siempre juega con las blancas – .

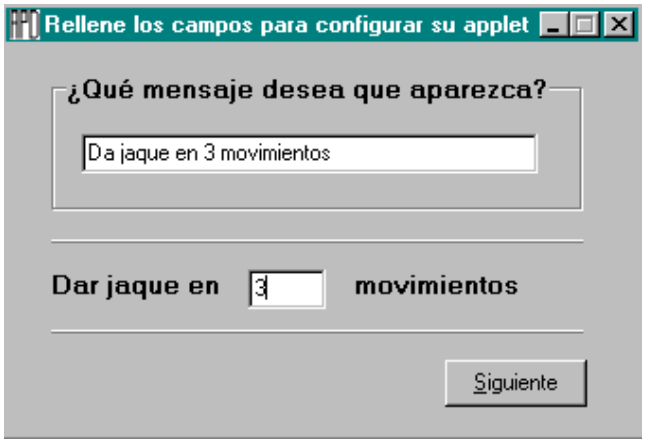
La herramienta avisará de posibles despistes como que falte alguno de los dos reyes, imprescindibles para llevar a cabo este tipo de ejercicios.

Una vez que se haya guardado correctamente la colocación de las piezas, se activa el botón que conducirá al usuario a las siguiente fase en la generación del ejercicio.

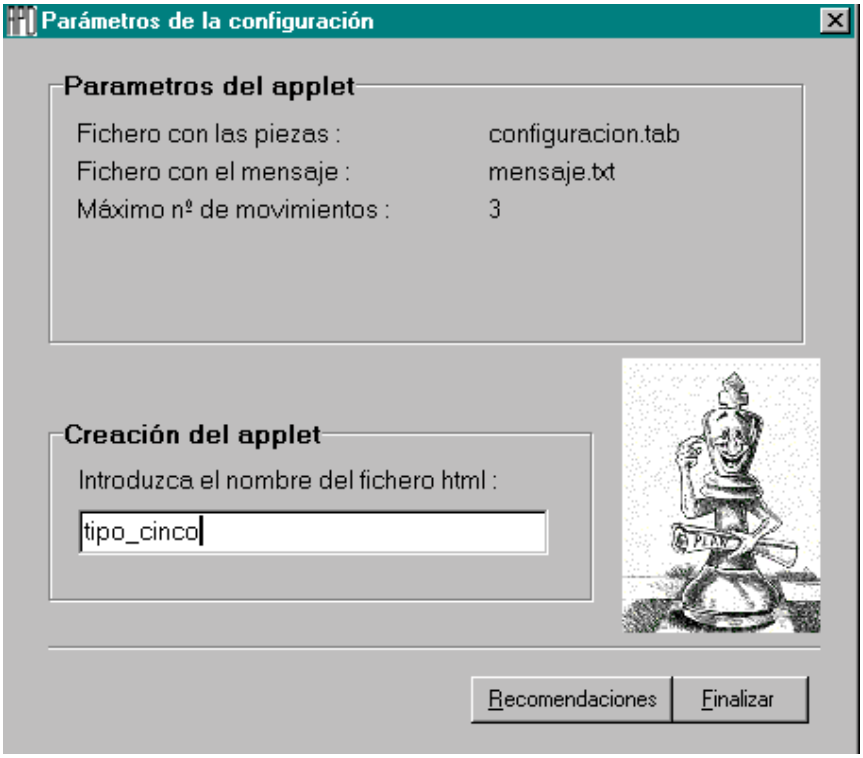




Se rellenarán los campos que aparecen en la siguiente pantalla: el mensaje que encabezará el ejercicio y el número máximo de movimientos que puede realizar el aprendiz para dar jaque al rey negro.



Finalmente, se muestran los parámetros de configuración del ejercicio y el nombre de la página html que lo albergará.



El ejercicio generado tiene el siguiente aspecto: