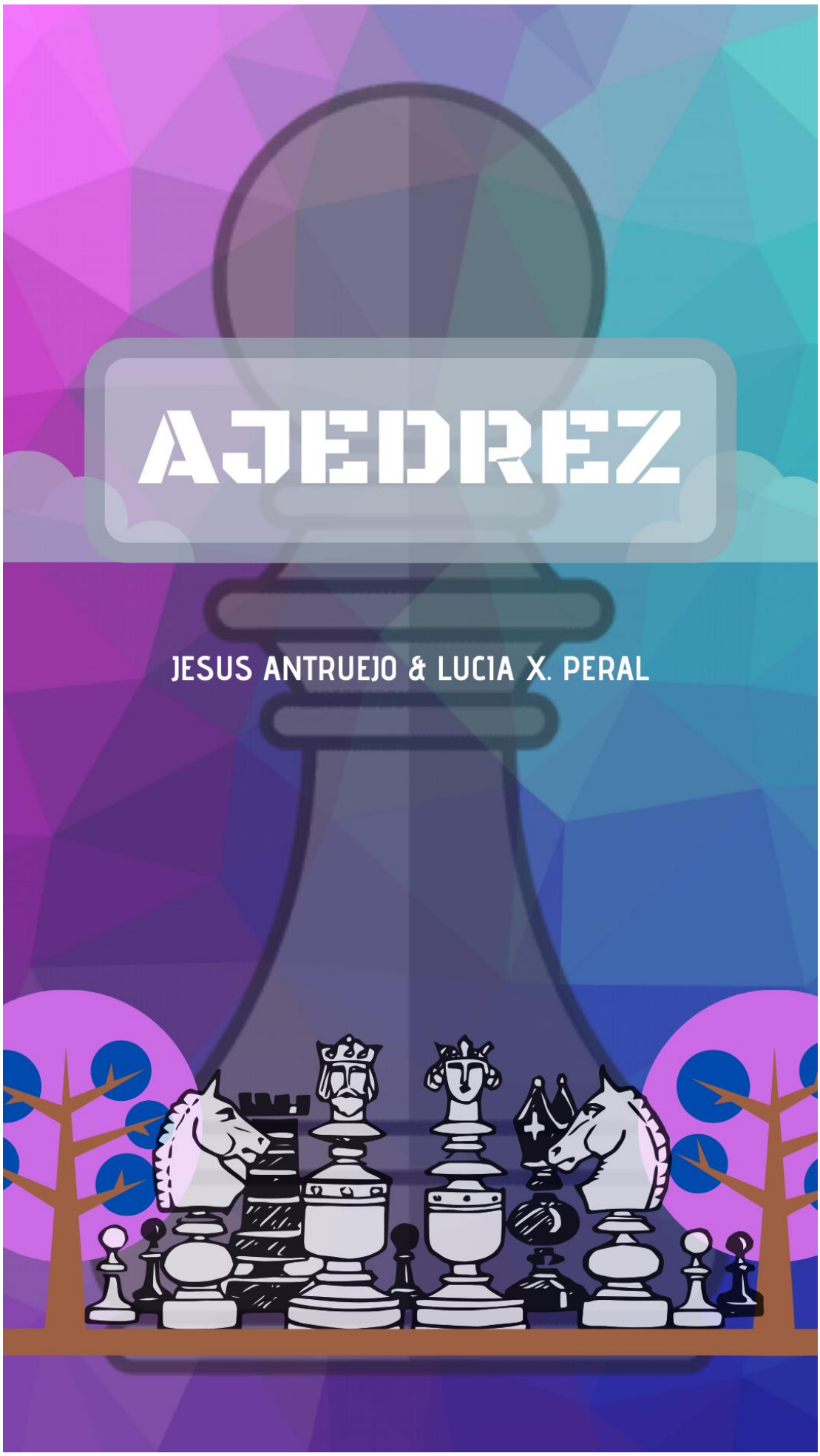
AJEDREZ

JESUS ANTRUEJO & LUCIA X. PERAL



Contenido

INTRODUCCIÓN

Hemos creado un juego básico de ajedrez que permite jugar a dos personas por turnos.

Para ello hemos usado el entorno de desarrollo de Netbeans y todas las funcionalidades que este permite.

Primero, hemos dividido en paquetes las piezas, el núcleo del programa y la interfaz gráfica.

A continuación, explicaremos en detalle todas las clases, métodos y objetos utilizados para lograr que el juego funcione correctamente.

PROPUESTA INICIAL

Al principio, habíamos pensado en hacer un juego básico de Ajedrez que permita jugar a dos personas por turnos.

La base de este Ajedrez consistiría en un array (probablemente multidimensional) que representara las coordenadas del tablero de juego.

Por encima trabajaría una clase abstracta "Ficha" de la que derivan cada tipo de ficha. Ficha debe de tener un campo color, y un campo coordenadas.

Para posicionar las fichas en el tablero, se crearía una instancia de cada una de las fichas (o varias si esto se requiere) y se "colocarían" en el tablero virtual que representa el array, asignándole las coordinadas apropiadas.

Partiendo del instanceof para sacar el tipo de ficha, las coordenadas y el campo color, podemos hacer las condiciones adecuadas para el movimiento y la eliminación de fichas.

La interfaz es algo que nos estamos planteando, tocaría investigar sobre qué clase de tablas se pueden meter en un Frame, y si alguna permite las siguientes características:

1) Inputs del usuario (para elegir ficha y elegir posición a la que mover)

2) Imágenes o iconos (para representar a las fichas)

ESTRUCTURA DEL PROGRAMA

Como dijimos previamente, dividimos en tres paquetes para organizar nuestro programa:

- PIEZAS

Este paquete se compone de la clase abstracta Pieza de la que deriva el resto de las fichas. Tiene los atributos Color e icono. Color sirve para decir si es blanca o negra la pieza, y viene de un Enum. El icono lo hemos escrito como String, ya que hemos usado las piezas de ajedrez del UNICODE. También hemos puesto los atributos x e y que son los números que representan la posición en la que se van a encontrar nuestras piezas.

Por último, creamos dos métodos abstractos para poder sobrescribirlos en las piezas.

Uno de ellos es para poder añadir el icono dependiendo de su color, y el otro es para restringir los movimientos y el comportamiento de las fichas.

Piezas que extienden de Pieza: Torre, Alfil, Caballo, Rey, Reina y Peón.

- NÚCLEO

Este paquete solo contiene la clase Cerebro. Esta clase es nuestro Main, y aquí se concentra la base funcional del programa. Antes de tener una Interfaz Gráfica, ya pudimos probar en el Main el funcionamiento de los métodos.

Además, aquí guardamos dos ArrayList de Pieza: el de fichas (que contiene todas las fichas al inicio de la partida) y el cementerio (que a medida que se van eliminando las fichas, se añaden al cementerio y se quitan de fichas).

Posteriormente en este documento, explicaremos con detalle los métodos que el Cerebro contiene.

- INTERFAZ GRÁFICA

Este paquete tiene la ventana de Inicio, el tablero y las dos imágenes usadas para que la Interfaz sea más bonita.

La ventana de Inicio se instancia en la clase Cerebro (que es la que tiene el Main). Y de la ventana de Inicio, podemos o bien ir a leer las reglas a una página que contiene un pdf de las leyes del ajedrez publicado por la FEDA (Federación Española de Ajedrez), o bien jugar.

Si le das al botón jugar, este te lleva a la ventana del tablero donde se instancian todos los métodos del cerebro para que funcione el juego de forma visual.

La ventana del Tablero es nuestro principal interfaz, y donde se desarrolla el juego. Tenemos un tablero en el que las casillas son botones (añadidos a un hashMap. Como clave un String de la posición que corresponde cada botón). Tenemos un botón de reiniciar (reinicia la partida), unos labels para dar información y un jTextArea donde salen las fichas que se guardan en el ArrayList de Cementerio. Por último, hay que decir que tiene los atributos piezaEnUso, turnoDe y turnos. Más adelante, explicaremos para que sirven.

FUNCIONAMIENTO DEL JUEGO

- RESTRICCIÓN DE MOVIMIENTOS: EN LAS PIEZAS

• PEÓN

El peón ha sido una de las piezas más difíciles de hacer funcionar. El peón normalmente solo se mueve una casilla hacia delante. Sin embargo, tiene tres restricciones. La primera es que, si aún no se ha movido el peón de su posición inicial, puede avanzar hacia adelante dos casillas. La segunda es que no puede retroceder. Y la última, es que, si tiene un enemigo en su diagonal, puede comérselo (no puede comer enemigos si se mueve hacia adelante, solo en diagonal y una casilla).

• TORRE

Esta pieza solo puede moverse horizontal o verticalmente. Para ello, o bien se suma/resta una cantidad a *X* y entonces *Y* se queda en 0 (horizontal) o viceversa (vertical).

• ALFIL

Esta pieza solo puede moverse en diagonal. La clave está en sumarle/restarle la misma cantidad a *X* que a *Y*.

• REY Y REINA

Hicimos ambos códigos muy parecidos, pues estos se pueden mover horizontal, vertical y diagonalmente. Para ello mezclamos los códigos escritos en la torre y el alfil.

Sin embargo, el rey puede hacer todos esos movimientos, pero solo moviéndose una casilla.

• CABALLO

Esta pieza es otra de las que se comportan de forma peculiar. Nos dimos cuenta de que o bien se suma/resta 2 a *X* y 1 a Y, o viceversa.

- RESTRICCIÓN DE MOVIMIENTOS: MÉTODOS

• MÉTODO MOVER FICHA

Este es uno de los métodos más complejos que hemos creado en el juego, ya que usa condiciones de otros métodos.

En primer lugar, comprueba que la ficha no esté fuera del tablero (**OutOfBounds**). Y después, si de camino a la casilla a la que quiere moverse hay una ficha, entonces no te deja mover la ficha. Si el camino está libre, comprueba otra condición en relación con el peón (pues este se comporta de forma diferente al resto). Hace un instanceof y si es un Peón y además hay un

enemigo a una casilla en diagonal de distancia, entonces comprueba si es o no enemigo (si es se lo come, sino no hace nada porque de normal no se puede mover en diagonal).

Si la ficha no es un Peón entonces, comprueba que esa pieza se pueda mover (**puedeMoverseA** método abstracto sobrescrito en todas las piezas) comprueba que la posición a la que quiere ir hay un enemigo, un aliado o no hay nadie. Si hay un enemigo, se lo come; si hay un aliado, no puede moverse y entonces debes moverla a otro sitio o mover otra pieza; si está vacía la posición y puedes moverte a ella, te mueves.

Para comprobar si el peón tiene a alguien en su diagonal, hemos creado un método (**hayEnemigoEnDiagonal** en Cerebro) que comprueba si hay un enemigo en la diagonal de la posición el la que se encuentra el peón.

Para comprobar si hay un enemigo, hay un método (**enemigoEnCoordenadas** en Cerebro) que recorre el ArrayList de las piezas y compara el color de la pieza que se quiere mover, con la de la pieza que te has encontrado en la posición a la que quieres ir. Devuelve false si los colores son iguales.

Lo mismo sucede con aliado, hay un método (**aliadoEnCoordenadas** en Cerebro) que funciona igual que enemigo. Aunque en este caso, devuelve true si los colores son iguales.

También este método usa un método (**encuentraFicha** en Cerebro) que te devuelve una pieza si encuentra dicha pieza en las coordenadas que tu quieras. (Te dice si en esas coordenadas hay o no una ficha).

- MÉTODO COLISIONADOR

Este método (**colisionador** de Cerebro) sirve para comprobar si hay una ficha en el camino. Imagínate que quieres ir de (2, 1) a (2, 5). Para ello tienes que comprobar si hay alguna ficha en el camino, porque ninguna ficha excepto el Caballo, puede volar. La función de este método es ir comprobando casilla por casilla desde el punto de origen hasta el de destino. En el ejemplo anterior, comprobaríamos si hay alguien en (2, 2), (2, 3) y (2, 4).

- RESTRICCIÓN LÍMITES TABLA

Para ello, hemos creado un método que comprueba dadas unas coordenadas, si estas están fuera del tablero. El tablero es de 8x8, aunque lo hemos numerado desde 0. (**OutOfBounds** en la clase Cerebro)

- VICTORIA EN LA PARTIDA

Hicimos un método que comprueba si hay un Rey en el cementerio. Este método se usa después en el Tablero como condición, para poder sacar el color del Rey que ha muerto y entonces poder proclamar ganador al otro color. (**winState** en la clase Cerebro)

- MÉTODOS DEL TABLERO

El tablero tiene métodos que usan de los métodos del Cerebro para que todas las condiciones funcionen.

En primer lugar, tenemos los métodos del turno. **ContadorTurnos**, lo hemos creado con el propósito de ir contando cada ronda, ya que, si llega a 50, hemos puesto que se acabe la partida en empate. **ActualizarTurno** lo hemos creado para que vaya diciendo a quién le toca jugar, si a las blancas o a las negras. Además, las fichas blancas siempre empiezan la partida, por lo que resultó más fácil programar los turnos.

En segundo lugar, tenemos el método principal para que funcione el resto de las cosas de la tabla.

Acción usa otros métodos como condiciones para que la ficha se mueva y haga lo que tiene que hacer. Recoge como parámetro el String coordenadas del botón. Cada botón tiene asignada la coordenada de su posición en un HashMap. Su clase es el String coordenadas del botón.

Primero recorre el ArrayList de las fichas y si las coordenadas del botón corresponden con la posición de la ficha que quieres mover, entonces, comprueba que sea el turno de la ficha seleccionada. Si no lo es, sale un mensaje de que es el turno del otro color y entonces no se mueve.

Si es el turno de tu color, la pieza comprueba que la ficha se puede mover a la posición de destino (**EncuentraFicha**), y si es así se mueve (**MueveFicha**)y refresca el turno (**ActualizarTurno**, **RefrescarContadorTurnos**) y el tablero (**refrescarTablero**). A parte, cada vez una pieza está en uso, hay un jLabel que te enseña cual es (**refrescarGUI**).

Si no se puede mover porque hay un enemigo, aliado o el movimiento que quieres hacer no es correcto para esa ficha, entonces salen los mensajes pertinentes.

También comprueba cada vez el **WinState** o si los turnos han llegado a 50. En el caso del **WinState**, la partida acaba y mira que color es el ganador. En el otro caso, la partida acaba en empate. Y después de que en cualquiera de los dos casos la partida acabe, se reinicia la ventana para jugar otra partida (**reiniciarPartida**).

Por último, tenemos el método de **reiniciarPartida**. Lo que hace este método es que, si la partida ha acabado, se cierra la ventana actual y llama al **reset** de Cerebro para generar una nueva ventana de Tablero.

RESUMEN DE OBJETOS, CLASES Y ELEMENTOS

- 1 Clase Main llamada Cerebro, que llama a la ventana de Inicio y contiene la mayoría de los métodos
- 2 jFrame: Inicio y Tablero (Inicio llama a Tablero con un botón)
- 1 URL en Inicio para que te lleve al pdf de la FEDA por siquieres leer las reglas
- 1 Clase abstracta de la que extienden 6 clases (Alfil, Caballo, etc) y con dos métodos abstractos sobrescritos en cada una de las clases herederas.
- 1 Enum de Color (Blanca y Negra) para el color de las fichas
- 2 ArrayList<Pieza>: fichas y cementerio

- 1 HashMap de los botones que hacen de casillas del tablero (64 botones)
- 2 imágenes para decorar
- 1 jTextArea para el cementerio
- Varios jLabels para brindar al jugador información de la partida
- Etc…

PRUEBAS PARA COMPROBAR EL FUNCIONAMIENTO

A lo largo del desarrollo hemos ido probando que funcionaban correctamente los métodos creados. Para ello, al principio cuando no teníamos interfaz gráfica, usábamos el Main y la consola de OUTPUT. Le metíamos la ficha que queríamos mover y la posición adonde queríamos llevarla. En los métodos habíamos escrito System.out.println para que saliera un mensaje por el OUTPUT y comprobar que la ficha "se movía" correctamente.

Después cuando tuvimos interfaz, simplemente movíamos las fichas y observábamos el comportamiento de estas. Comprobábamos en cada tipo de ficha, si podía moverse como su tipo se suele mover (ejemplo: el alfil solo se mueve en diagonal). O si intentábamos llevarla de un punto a otro y había una ficha en el medio, tenía que salir un mensaje de que no se podía mover ahí.

En el vídeo de demostración, se puede visualizar la mayoría de las pruebas que hemos ido haciendo.

CONCLUSIÓN

Estamos satisfechos con el trabajo realizado. A pesar de que hemos tenido dificultades con la interfaz gráfica al principio porque no teníamos ni idea de como hacerlo, hemos salido adelante y creemos que nos ha quedado bastante bonita, visualmente.

Es cierto, que nuestro proyecto tiene algunas desventajas. Como por ejemplo que funcione lento por la enorme cantidad de código y mala optimización de este. Además, si das órdenes al programa muy rápido no responde bien por lo dicho antes de la optimización.

También que no hayamos podido implementar algunas funciones del ajedrez como el enroque o los jaques.

Sin embargo, con este trabajo hemos aprendido muchas cosas, y además hemos logrado que el juego funciona de forma intermedia (ya no básica) y para ello hemos aplicado solo los conocimientos dados en clase.

También hemos aprendido a resolver problemas complejos de poco en poco para que todo se coordine, es decir, entrelazar varios métodos para que juntos hagan una función entera con todas las restricciones necesarias.