IMPLEMENTACIÓN DEL JUEGO DEL AJEDREZ

UNIVERSIDAD DE LAS ISLAS BALEARES
TECNOLOGÍA DE LA PROGRAMACIÓN
2006 / 2007

Alumno: Ligia Tatiana González Leyva {X4741315A}

Índice de contenido

1. Introducción

La aplicación a desarrollar consiste en crear un programa en Java que simule el juego del ajedrez; en concreto deberá permitir desarrollar una partida humano vs. computadora.

2. Simplificaciones

Durante el desarrollo del ajedrez y debido principalmente a la cantidad de aspectos que puede abarcar una aplicación de ajedrez real y también a que ha sido desarrollada por una sola persona se han tomado una serie de simplificaciones que no afectan, sin embargo, al objetivo central de la práctica que consiste en la aplicación de técnicas de backtracking para solventar las jugadas a realizar por parte de la computadora. Sin más tales simplificaciones se listan a continuación:

En cuánto al propio juego en sí:

- No se han considerado las jugadas de jaque y jaque mate. La partida acaba cuándo uno de los jugadores consigue "matar" al rey del oponente (regla no válida en la realidad pero sí aproximada).
- Un jugador no puede solicitar tablas. Únicamente se produce una situación de tablas cuándo el usuario no puede realizar ningún movimiento.

En cuánto al jugador humano:

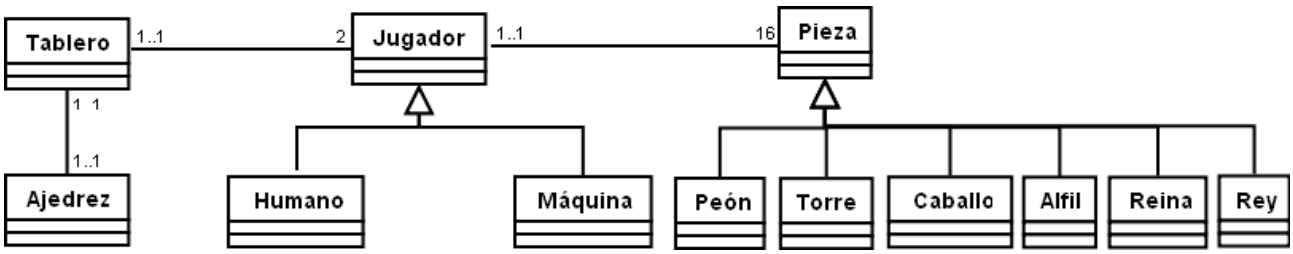
- Siempre jugará con las piezas de color blanco.

En cuánto al jugador máquina:

- Siempre jugará con las piezas de color negro.
- Su inteligencia vendrá limitada por el proceso que se explicará más adelante.

3. Estructuración de la aplicación

A continuación se muestra un modelo con las clases que contiene la aplicación:



3.1. Ajedrez

Ésta es la clase principal de la aplicación y será la que sea invocada para iniciar la aplicación.

Es la responsable de la interfaz gráfica de usuario y por ello es una subclase de Jframe.

Contiene los siguientes métodos importantes:

- Main: punto de entrada de la aplicación crea un nuevo objeto de tipo Ajedrez.
- Constructor: fija los atributos de la ventana creada creando todos los elementos de juego y controlando el propio desarrollo del juego.

3.2. Tablero

Esta clase controla la situación real de la partida matiendo constancia de la situación de las fichas de cada uno de los jugadores.

Para ello dispone de dos arrays uno de las propias piezas y uno de Casillas, siendo **Casilla** una clase que hereda de Jpanel y que contiene un Jlabel; su función principal es poder dibujar un icono de la ficha correspondiente en la pantalla.

Volviendo a la clase tablero tenemos los siguientes métodos principales:

- add: añade una ficha al tablero.
- Get: devuelve que ficha se halla en una determinada fila y columna.
- MostrarGUI: muestra gráficamente la situación actual del tablero.
- GenerarMovimientos: genera los movimientos posibles para un color de juego determinado (blanco, negro).
- ValorTablero: retorna la valoración actual de un tablero para un color determinado teniendo en cuenta que las fichas propias suman mientras que las rivales restan. Cada ficha tiene un valor asignado en función de su importancia. Un valor de 0 indicará una situación de empate.

3.3. Jugador

Clase base abstracta de la que heredarán los dos tipos de jugadores existentes en la aplicación: humanos y máquina.

Cómo métodos no abstractos importantes destacan:

- realizarJugada: invocable con parámetros diferentes; en su primera versión permite mover de una coordenada inicial a una coordenada final en el tablero que tiene asignado el jugador al iniciar la partida. En su segunda versión se utiliza para evaluar jugadas y toma cómo parámetros un tablero y un movimiento.

- HaPerdido: un jugador pierde cuándo el estado de su rey es “comido”.

En cuanto a los métodos abstractos a redefinir el más importante es mover.

3.4. Jugador Humano

Corresponde al jugador controlado por el usuario.

Centrándonos en la redefinición de su método mover:

- Se despliega un panel de opción al usuario dónde este introduce la jugada a realizar mediante las coordenadas respectivas. Una vez verificadas las coordenadas introducidas se verifica la jugada invocando al método jugadaValida que comprueba básicamente que la posición destino esté libre u ocupada por un enemigo y a continuación verifica que la pieza puede realizar ese movimiento invocando al método validarMovimiento de la clase Pieza.

3.5. Jugador Máquina

Corresponde al jugador controlado por la máquina.

Su método **mover** es diferente al anterior:

- Primero se calculan los movimientos posibles a realizar sobre el tablero.
- Para cada uno de los movimientos calculados se efectúa sobre un tablero auxiliar y a continuación se invoca a la función buscar.

La función **buscar** funciona cómo se describe a continuación:

- Toma cómo argumento un tablero y una profundidad.
- Si la profundidad es cero devuelve la evaluación del tablero. (caso base).
- Si no genera todos los movimientos posibles para el tablero introducido y los evalúa rellamándose a si misma con un nivel de profundidad menos. De todos los valores devueltos por los movimientos que se generaron se toma el más alto siendo ese el que se retorna finalmente.
- El resultado final es que se devuelve la puntuación máxima de las jugadas que pueden realizarse.

Volviendo al método **mover** se confeccionará una lista con aquellos movimientos que tengan mayor puntuación (mejores). Una vez obtenida dicha lista se procedera a priorizar aquellos que impliquen matar a una ficha rival en ese mismo movimiento (puede que dos movimientos tengan el mismo valor ya que uno mata en primera instancia a una ficha rival pero el otro lo hace en dos jugadas).

Finalmente para todos los movimientos con prioridad se seleccionará uno aleatoriamente.

3.6. Pieza

Clase abstracta que representa las fichas que intervienen en el juego.

Una pieza podrá pertenecer únicamente a un jugador.

Al tener representación gráfica tendrán un icono asociado.

3.7. Peón, Torre, Caballo, Alfíl, Reina, Rey

Heredan directamente de la clase Pieza y redefinen los métodos que se explican a continuación:

- ValidarMovimiento: para un tablero y una posición destino comprueban si pueden efectuar ese movimiento en función de la pieza a la que estén representando (por ejemplo un peón no podrá saltar a otra ficha).
- PosiblesMovimientos: para un tablero generan una lista de movimientos posibles en función de su posición actual.

3.8. Constantes

Es una interfaz que contiene cómo su nombre indica las constantes utilizadas por la aplicación, por ejemplo los colores.

3.9. Movimiento

Clase que representa un movimiento sobre el juego. Para ello contiene información de una pieza a mover y una posición final sobre la que moverse.

4. Ejecución de la aplicación

Para ejecutar la aplicación se adjunta el script **run.bat** que lo único que hace es compilar y ejecutar la clase Ajedrez.

Puede que sea necesario modificarlo para adaptarlo a un nuevo computador destino ya que en ordenador sobre el que se desarrolló la aplicación era necesario definir en el path del sistema la ruta del compilador de java cada vez.

5. Código Fuente

5.1. Ajedrez.java

```
/*
 *$LastChangedDate: 2007-09-18 20:31:00 +0200 (mar, 18 sep 2007) $
```

```
 *$LastChangedRevision: 38 $
 * Ligia Tatiana Gonzalez Leyva <calvinahobbes@gmail.com>
 */

import java.io.*;
import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;

public class Ajedrez extends JFrame implements Constantes{

	Tablero t;
	Jugador jH;
	Jugador jM;

	public Ajedrez () {

	this.setSize(500,500);
	this.setTitle("Ajedrez 1.0");
		//this.setVisible(true);
		this.addWindowListener( new WindowAdapter()
			{
				public void windowClosing(WindowEvent e)
				{
					dispose();
					System.exit(0);
				}
			}
		);

		// crear tablero y jugadores
		t = new Tablero(this);
		jH = new JugadorHumano("Tatiana Gonzalez", BLANCO, t);
		jM = new JugadorMaquina("CPU", NEGRO, t);

		Container cp = getContentPane();
	cp.setLayout(new FlowLayout());

		t.crearTableroGUI(this);
		t.mostrarGUI();

		if (DEBUG)
			t.mostrar();
		show();
```

```
                        // desarrollo de la partida
                        while (!jH.haPerdido() && !jM.haPerdido() && !t.getTablas()) {

                                jH.mover();
                                jM.mover();
                        }

                        if (t.getTablas())
                                JOptionPane.showMessageDialog(null, "La partida ha acabado
en tablas");
                        else if (jH.haPerdido())
                                JOptionPane.showMessageDialog(null, jM.getNombre()+", ha
ganado!!");
                        else
                                JOptionPane.showMessageDialog(null, jH.getNombre()+", ha
ganado!!");

                        this.dispose();
                }

                public static void main(String[] args) {

                        System.out.println("Ajedrez.main(): Arrancando GUI");
                        new Ajedrez();
                }
}
```

5.2. Alfil.java

```
/*
 *$LastChangedDate: 2007-09-18 01:28:13 +0200 (mar, 18 sep 2007) $
 *$LastChangedRevision: 29 $
 * Ligia Tatiana Gonzalez Leyva <calvinahobbes@gmail.com>
 */

import java.util.*;
import java.io.*;

public class Alfil extends Pieza {

                // Constructor
                public Alfil (int color, int x, int y) {

                        super(color, x, y, "imagenes/a");
```

```
                        //System.out.println("Alfil.Alfil(): \t\tCreado con color <" + color + "> y
posicion (" + x + ", " + y + ")");
                }

                public String toString () {

                        if (color == BLANCO)
                                return "A";
                        else
                                return "a";
                }

                public Pieza copiar () {

                        return new Alfil(this.color, this.f, this.c);
                }

                public boolean validarMovimiento (Tablero tablero, int fF, int cF) {

                        if (fF > 7 || fF < 0)
                                return false;
                        if (cF > 7 || cF < 0)
                                return false;

                        // diferencia entre filas y columnas para el movimiento
                        int difFil, difCol;

                        Pieza p;

                        p = tablero.get(fF, cF);

                        difFil = Math.abs(this.f - fF);
                        difCol = Math.abs(this.c - cF);

                        //un valor igual para ambas diferencias indica un moviento en diagonal
                        if ( (difFil == difCol) ) {
                                // la casilla destino no esta vacia
                                if (p != null )
                                        //no se puede comer una pieza de igual color
                                        if (p.getColor() == this.color)
                                                return false;

                                // el alfil no puede saltar a otras fichas
                                for ( int i = 1; i < difFil; i++ ) {
                                        p = tablero.get( this.f + i * ((fF - this.f) / difFil),
this.c + i * ((cF - this.c) / difFil));
```

```
                                        // trata de saltar a otra ficha
                                        if (p != null )
                                                return false;
                                }
                                return true;
                        }
                        // no es un movimiento en diagonal
                        else
                                return false;
        }

        public ArrayList posiblesMovimientos(Tablero t) {

                ArrayList a = new ArrayList();

                for (int df=-7; df<=7; df++)
                        if (df != 0)
                                for (int dc=-7; dc<=7; dc++)
                                        if (dc != 0)
                                                if (validarMovimiento(t,
this.f+df, this.c+dc)) {
                                                        a.add(new
Movimiento(this, this.f+df, this.c+dc));

        //System.out.println("Alfil.posiblesMovimientos(): De <"+this.f+","+this.c+"> a
<"+(this.f+df)+","+(this.c+dc)+">");
                                                }
                        return a;
        }
}
```

5.3. Caballo.java

```
/*
 *$LastChangedDate: 2007-09-18 18:53:18 +0200 (mar, 18 sep 2007) $
 *$LastChangedRevision: 34 $
 * Ligia Tatiana Gonzalez Leyva <calvinahobbes@gmail.com>
 */

import java.util.*;
import java.io.*;

public class Caballo extends Pieza {

        // Constructor
```

```
	public Caballo (int color, int x, int y) {

		super(color, x, y, "imagenes/c");
		//System.out.println("Caballo.Caballo(): \tCreado con color <" + color + ">
y posicion (" + x + ", " + y + ")");
	}

	public String toString () {

		if (color == BLANCO)
			return "C";
		else
			return "c";
	}

	public Pieza copiar () {

		return new Caballo(this.color, this.f, this.c);
	}

	public boolean validarMovimiento (Tablero tablero, int fF, int cF) {

		if (fF > 7 || fF < 0)
			return false;
		if (cF > 7 || cF < 0)
			return false;

		int difFil, difCol;
		Pieza p;

		p = tablero.get(fF, cF);

		difFil = this.f - fF;
		difCol = this.c - cF;

		// el movimiento del caballo se caracteriza por un salto de
		// 1 en las filas y 2 en las columnas o viceversa (en valor absoluto)
		if (((Math.abs(difFil) == 1) && (Math.abs(difCol) == 2)) ||
((Math.abs(difFil) == 2) && (Math.abs(difCol) == 1))) {
			// solo podemos comer al rival
			if (p != null )
				if (p.getColor() == this.color)
					return false;
			return true;
		}
		else
```

```
                                        return false;
                }

                public ArrayList posiblesMovimientos(Tablero t) {

                        ArrayList a = new ArrayList();

                        for (int df=-2; df<=2; df++)
                                if (df != 0)
                                        for (int dc=-2; dc<=2; dc++)
                                                if (dc != 0)
                                                        if (validarMovimiento(t,
this.f+df, this.c+dc)) {
                                                                a.add(new
Movimiento(this, this.f+df, this.c+dc));
                                                        }
                        return a;
                }
}
```

5.4. Constantes.java

```
/*
 *$LastChangedDate: 2007-09-18 20:31:00 +0200 (mar, 18 sep 2007) $
 *$LastChangedRevision: 38 $
 * Ligia Tatiana Gonzalez Leyva <calvinahobbes@gmail.com>
 */

//import java.io.*;

public interface Constantes {

        // colores
        static int BLANCO      = 1;
        static int NEGRO       = 0;

        // profundidad de la busqueda del jugador maquina
        static int MAXPROF     = 1;

        // debug
        static boolean DEBUG = false;
}
```

5.5. Jugador.java

```
/*
```

```
 *$LastChangedDate: 2007-09-18 02:16:51 +0200 (mar, 18 sep 2007) $
 *$LastChangedRevision: 30 $
 * Ligia Tatiana Gonzalez Leyva <calvinahobbes@gmail.com>
 */

import java.io.*;
import java.util.*;

public abstract class Jugador implements Constantes {

	// nombre del jugador
	String nombre;
	// color con el que juega el jugador
	int color;
	// piezas del jugador
	ArrayList piezas;
	// tablero de juego
	Tablero tablero;

	// constructor
	public Jugador (String nombre, int color, Tablero t) {

		this.nombre = nombre;
		this.color = color;
		piezas = new ArrayList();
		tablero = t;

		int posicion;
		if (color == BLANCO)
			posicion = 1;
		else
			posicion = 0;

		// Crear las piezas y ponerlas en el tablero
		Pieza p;
		// crear peones
		for (int i=0; i<8; i++) {
			p = new Peon(color, 5*posicion+1, i);
			piezas.add(p);
			tablero.add(5*posicion+1, i, p);
		}
		// crear torres
		for (int i=0; i<2; i++) {
			p = new Torre(color, 7*posicion, 7*i);
			piezas.add(p);
			tablero.add(7*posicion, 7*i, p);
```

```
                        }
                        // crear caballos
                        for (int i=0; i<2; i++) {
                                p = new Caballo(color, 7*posicion, 5*i+1);
                                piezas.add(p);
                                tablero.add(7*posicion, 5*i+1, p);
                        }
                        // crear alfiles
                        for (int i=0; i<2; i++) {
                                p = new Alfil(color, 7*posicion, 3*i+2);
                                piezas.add(p);
                                tablero.add(7*posicion, 3*i+2, p);
                        }
                        // crear reina
                        p = new Reina(color, 7*posicion, 4);
                        piezas.add(p);
                        tablero.add(7*posicion, 4, p);
                        // crear rey
                        p = new Rey(color, 7*posicion, 3);
                        piezas.add(p);
                        tablero.add(7*posicion, 3, p);

                        //System.out.println("Jugador.Jugador(): Me acaban de crear");
        }

        public abstract void mover ();

        public void realizarJugada (int fI, int cI, int fF, int cF) {

                        Pieza pAMover, pAComer;
                        boolean bResultado;

                        //pieza a mover
                        pAMover = tablero.get( fI, cI );
                        //System.out.println("Jugador.realizarJugada(): mover de <"+fI+","+cI+"> a
<"+fF+","+cF+">");
                        //System.out.println("Jugador.realizarJugada(): la pieza a mover es " +
pAMover.toString());

                        //ficha a comer (si hay)
                        pAComer = tablero.get( fF, cF );

                        // hay que comer una pieza
                        if (pAComer != null)
                                pAComer.setEstado(true);
```

```
                        if (pAMover != null) {
                                tablero.add(fF, cF, pAMover);
                                tablero.add(fI, cI, null);
                                // nueva posicion de la pieza
                                pAMover.setPosicion(fF, cF);
                        }
                        tablero.mostrarGUI();
                        //tablero.mostrar();
                }

                // Realiza la jugada sobre un tablero diferente al asignado a la partida
                //
                public void realizarJugada (Tablero t, Movimiento m) {

                        int fI = (m.getPieza()).getFila();
                        int cI = (m.getPieza()).getColumna();
                        int fF = m.getFilaFinal();
                        int cF = m.getColumnaFinal();

                        Pieza pAMover, pAComer;
                        boolean bResultado;

                        //pieza a mover
                        pAMover = t.get( fI, cI );
                        //System.out.println("Jugador.realizarJugada(): mover de <"+fI+","+cI+"> a
<"+fF+","+cF+">");
                        //System.out.println("Jugador.realizarJugada(): la pieza a mover es " +
pAMover.toString());

                        //ficha a comer (si hay)
                        pAComer = t.get( fF, cF );

                        // hay que comer una pieza
                        if (pAComer != null)
                                pAComer.setEstado(true);

                        if (pAMover != null) {
                                t.add(fF, cF, pAMover);
                                t.add(fI, cI, null);
                                // nueva posicion de la pieza
                                pAMover.setPosicion(fF, cF);
                        }
                }

                public String getNombre () {
```

```
                return nombre;
        }

        // Comprueba si el rey del jugador no ha sido comido
        //
        public boolean haPerdido () {

                return ((Pieza)piezas.get(15)).getEstado();
        }
}
```

5.6. JugadorHumano.java

```
/*
 *$LastChangedDate: 2007-09-18 01:28:13 +0200 (mar, 18 sep 2007) $
 *$LastChangedRevision: 29 $
 * Ligia Tatiana Gonzalez Leyva <calvinahobbes@gmail.com>
 */

import java.io.*;
import java.util.*;
import javax.swing.event.*;
import javax.swing.*;

public class JugadorHumano extends Jugador {

        // Constructor
        public JugadorHumano (String nombre, int color, Tablero t) {

                super(nombre, color, t);
                //System.out.println("JugadorHumano.JugadorHumano(): " + nombre + "
creado.");
        }

        public void mover () {

                int fI = 0;
                int cI = 0;
                int fF = 0;
                int cF = 0;
                boolean valido = false;
                boolean direccionValida;
                String movimiento; // = JOptionPane.showInputDialog("Movimiento:\nEj:
A2 A4");
                String titulo = new String("Movimiento de "+ this.nombre + ", color "
```

```
+(this.color==BLANCO?"BLANCO":"NEGRO")+":\nEj: A7 A5");

                        while (!valido) {

                                movimiento = JOptionPane.showInputDialog(titulo);
                                while (movimiento == null || movimiento.equals("") ) {
                                        JOptionPane.showMessageDialog( null,
this.nombre+", introduce un movimiento." );
                                        movimiento =
JOptionPane.showInputDialog(titulo);
                                }

                                // validar formato del movimiento introducido
                                movimiento = movimiento.toUpperCase();
                                direccionValida = true;
                                if (movimiento.length() != 5) {
                                        JOptionPane.showMessageDialog( null,
this.nombre+", formato de direccion invalido." );
                                        direccionValida = false;
                                }
                                if (direccionValida && ((movimiento.charAt(0) < 'A' ||
movimiento.charAt(0) > 'H') || (!Character.isDigit(movimiento.charAt(1))))) {
                                        JOptionPane.showMessageDialog( null,
this.nombre+", formato de la direccion de inicio invalido" );
                                        direccionValida = false;
                                }
                                if (direccionValida && (movimiento.charAt(2) != ' ')) {
                                        JOptionPane.showMessageDialog( null,
this.nombre+", formato de direccion invalido." );
                                        direccionValida = false;
                                }
                                if (direccionValida && ((movimiento.charAt(3) < 'A' ||
movimiento.charAt(3) > 'H') || (!Character.isDigit(movimiento.charAt(4))))) {
                                        JOptionPane.showMessageDialog( null,
this.nombre+", formato de la direccion de fin invalido." );
                                        direccionValida = false;
                                }
                                if (direccionValida) {
                                        fI = Integer.parseInt(movimiento.charAt(1)+"") - 1;
                                        cI = movimiento.charAt(0) - 'A';
                                        fF = Integer.parseInt(movimiento.charAt(4)+"") - 1;
                                        cF = movimiento.charAt(3) - 'A';

                                        valido = jugadaValida(fI, cI, fF, cF);
                                        if (!valido)
                                                JOptionPane.showMessageDialog( null,
```

```
this.nombre+", ese movimiento es invalido" );
                                        }
                        }

                        realizarJugada(fI, cI, fF, cF);
        }

        public boolean jugadaValida (int fI, int cI, int fF, int cF) {

                        Pieza p, fFin;
                        //Cuadro c;

                        p = tablero.get(fI, cI);

                        // no hay ficha
                        if ( p == null )
                                return false;

                        //no es el color que esta jugando
                        if ( p.getColor() != this.color )
                                return false;

                        return p.validarMovimiento(tablero, fF, cF);
        }
}
```

5.7. JugadorMaquina.java

```
/*
 *$LastChangedDate: 2007-09-18 20:31:00 +0200 (mar, 18 sep 2007) $
 *$LastChangedRevision: 38 $
 * Ligia Tatiana Gonzalez Leyva <calvinahobbes@gmail.com>
 */

import java.util.*;
import java.io.*;

public class JugadorMaquina extends Jugador {

        // Constructor
        public JugadorMaquina (String nombre, int color, Tablero t) {

                        super(nombre, color, t);
                        //System.out.println("JugadorMaquina.JugadorMaquina(): Me acaban de
crear");
```

```
	}

	public void mover () {

		int aux = 0;
		// al inicio de la jugada tenemos un valor de tablero que intentaremos
superar
		int valorActual = this.tablero.valorTablero(this.color);
		// mejores movimientos que se van a calcular
		//ArrayList mejores=new ArrayList();
		ArrayList movimientos = this.tablero.generarMovimientos(this.color);
		ArrayList mejores = new ArrayList();

		Iterator it = movimientos.iterator();
		while(it.hasNext()) {
			Tablero tNuevo = new Tablero(this.tablero);
			Movimiento mov = (Movimiento)(it.next());
			if (DEBUG)
				System.out.println("JugadorMaquina.mover():
"+mov.toString()+"; valor del nodo: "+tNuevo.valorTablero(this.color));
			realizarJugada(tNuevo, mov);
			aux = buscar(tNuevo, MAXPROF);

			if (aux > valorActual) {
				valorActual = aux;
				mejores = new ArrayList();
				mejores.add(mov);
			}
			else if (aux == valorActual)
				mejores.add(mov);
		}

		// si el jugador no tiene posibilidad de moverse
		// se decreta la partida en tablas
		if (mejores.size() == 0)
			this.tablero.setTablas();
		else {
			// entre las mejores jugadas seleccionadas, prioriza aquellas que
implican comer otra ficha
			ArrayList optimizacion = new ArrayList();
			it = mejores.iterator();
			Movimiento mov;
			while(it.hasNext()) {
				mov = (Movimiento)(it.next());
				if (DEBUG)
```

```
            System.out.println("JugadorMaquina.mover(): movimiento: "+mov.toString());
                                                // la jugada implica comer a otra ficha
                                                if (this.tablero.get(mov.getFilaFinal(),
mov.getColumnaFinal()) != null)
                                                                optimizacion.add(mov);
                                    }
                                    if (DEBUG)
                                                System.out.println("JugadorMaquina.mover(): hay
"+optimizacion.size()+" jugadas optimizadas de "+mejores.size()+" posibles.");

                                    // hay jugadas que comen otras fichas
                                    if (optimizacion.size() != 0)
                                                mov = (Movimiento)(optimizacion.get((int)
(Math.random() * optimizacion.size())));
                                    else
                                                mov = (Movimiento)(mejores.get((int)
(Math.random() * mejores.size())));

                                    // sobre el tablero de la partida
                                    if (DEBUG)
                                                System.out.println("JugadorMaquina.mover():
movimiento elegido: "+mov.toString());
                                    realizarJugada((mov.getPieza()).getFila(),
(mov.getPieza()).getColumna(), mov.getFilaFinal(), mov.getColumnaFinal());
                        }
            }

            // Dada una situacion de tablero retorna la puntuacion maxima de las posibles jugadas
que pueden
            // realizarse.
            //
            public int buscar (Tablero t, int profundidad) {

                        // valorActual es el maximo valor hallado a partir del nodo
                        int valorActual = t.valorTablero(this.color);;
                        // evaluador de los subarboles del nodo
                        int valor = 0;

                        // limite, devolvemos el valor del tablero
                        if (profundidad == 0) {
                                    //int temp = t.valorTablero(this.color);
                                    if (DEBUG)
                                                System.out.println("JugadorMaquina.buscar():
Puntaje: "+valorActual+"  PROF: "+profundidad);
                                    return valorActual;
                        }
```

```
                        ArrayList movimientos = t.generarMovimientos(this.color);

                        // Evalua cada movimiento posible
                        Iterator it = movimientos.iterator();
                        while (it.hasNext()) {
                                Tablero tNuevo = new Tablero(t);
                                Movimiento mov = (Movimiento)(it.next());
                                //System.out.println("JugadorMaquina.buscar():
"+mov.toString()+" PROF: "+profundidad);
                        realizarJugada(tNuevo, mov);
                                valor = buscar(tNuevo, profundidad-1);
                                if (valor > valorActual)
                                        valorActual = valor;
                        }
                        if (DEBUG)
                                System.out.println("JugadorMaquina.buscar(): Puntaje devuelto:
"+valorActual+" PROF: "+profundidad);
                        return valorActual;
                }
}
```

5.8. Movimiento.java

```
/*
 *$LastChangedDate: 2007-09-18 02:16:51 +0200 (mar, 18 sep 2007) $
 *$LastChangedRevision: 30 $
 * Ligia Tatiana Gonzalez Leyva <calvinahobbes@gmail.com>
 */

import java.io.*;

public class Movimiento {

        Pieza p;
        int fF;
        int cF;

        // Constructor
        public Movimiento (Pieza p, int fF, int cF) {

                this.p = p;
                this.fF = fF;
                this.cF = cF;
        }
```

```
		public Pieza getPieza () {

			return p;
		}

		public int getFilaFinal () {

			return fF;
		}

		public int getColumnaFinal () {

			return cF;
		}

		public String toString () {

			return "<"+p.toString()+"> de <"+p.getFila()+","+p.getColumna()+"> a
<"+fF+","+cF+">";
		}
}
```

5.9. Peon.java

```
/*
 *$LastChangedDate: 2007-09-18 01:28:13 +0200 (mar, 18 sep 2007) $
 *$LastChangedRevision: 29 $
 * Ligia Tatiana Gonzalez Leyva <calvinahobbes@gmail.com>
 */

import java.util.*;
import java.io.*;

public class Peon extends Pieza {

		// Constructor
		public Peon (int color, int x, int y) {

			super(color, x, y, "imagenes/p");

			//System.out.println("Peon.Peon(): \t\tCreado con color <" + color + "> y
posicion (" + x + ", " + y + ")");
		}
```

```
	public String toString () {

		if (color == BLANCO)
			return "P";
		else
			return "p";
	}

	public Pieza copiar () {

		return new Peon(this.color, this.f, this.c);
	}

	public boolean validarMovimiento (Tablero tablero, int fF, int cF) {

		if (fF > 7 || fF < 0)
			return false;
		if (cF > 7 || cF < 0)
			return false;

		Pieza p;

		// valores para el movimiento de la pieza
		int incremento = (this.color==BLANCO)?-1:1;
		int filaSalida = (this.color==BLANCO)?6:1;

		p = tablero.get(fF, cF);

		//System.out.println("Peon.validarMovimiento(): posicion actual del peon
<"+this.f+","+this.c+">");
		//System.out.println("Peon.validarMovimiento(): lo movemos a
<"+fF+","+cF+">");

		// movimiento normal de los peones
		if ( (this.c == cF) && (this.f + incremento == fF) ) {
			// no puede comer asi
			if ( p != null )
				return false;
			return true;
		}
		//movimiento inicial de los peones
		else if ( (this.c == cF) && (this.f == filaSalida) && (fF == filaSalida +
2*incremento) ) {
			// no puede comer asi
			if ( p != null )
				return false;
```

```
                                        // verificamos que no haya una ficha en la mitad
                                        p = tablero.get(this.f + incremento, this.c);
                                        // no puede saltar a otra ficha
                                        if ( p != null )
                                                return false;
                                        return true;
                                }
                                //movimiento de comer
                                else if ( ((cF == this.c + 1) || (cF == this.c - 1)) && (fF == this.f +
incremento) ) {
                                        // no hay nada que comer
                                        if ( p == null )
                                                return false;
                                        // no es una ficha rival
                                        if ( p.getColor() == this.color )
                                                return false;
                                        return true;
                                }

                                return false;
                        }

   public ArrayList posiblesMovimientos(Tablero t) {

                        ArrayList a = new ArrayList();

                        for (int df=-2; df<=2; df++)
                                if (df != 0)
                                        for (int dc=-1; dc<=1; dc++)
                                                if (validarMovimiento(t, this.f+df, this.c+dc)) {
                                                        a.add(new Movimiento(this, this.f+df,
this.c+dc));

                        //System.out.println("Peon.posiblesMovimientos(): De <"+this.f+","+this.c+"> a
<"+(this.f+df)+","+(this.c+dc)+">");
                                                }
                        return a;
   }
}
```

5.10. Pieza.java

```
/*
 *$LastChangedDate: 2007-09-18 20:31:00 +0200 (mar, 18 sep 2007) $
```

```
 *$LastChangedRevision: 38 $
 * Ligia Tatiana Gonzalez Leyva <calvinahobbes@gmail.com>
 */

import java.util.*;
import javax.swing.*;
import java.io.*;

public abstract class Pieza implements Constantes {

		// estado de la pieza en la partida
		boolean comida = false;
		// color de la pieza
		int color;
		// posicion de la pieza en el tablero de juego
		int f, c;
		ImageIcon icono;

		// constructor
		// string contiene la ruta parcial al icono que se completa en funcion de su color
		//
		public Pieza (int color, int x, int y, String ruta) {

			this.color = color;
			f = x;
			c = y;
			if (color == BLANCO)
				icono = new ImageIcon(ruta+"b.gif");
			else
				icono = new ImageIcon(ruta+"n.gif");

			//System.out.println("Pieza.Pieza(): Me acaban de crear");
		}

		public ImageIcon getImagen () {

			return icono;
		}

		public abstract String toString ();

		public int getColor () {

			return color;
		}
```

```
	public void setPosicion (int f, int c) {

		this.f = f;
		this.c = c;
	}

	public int getFila () {

		return f;
	}

	public int getColumna () {

		return c;
	}

	public void setEstado (boolean comida) {

		this.comida = comida;
	}

	public boolean getEstado () {

		return comida;
	}

	// Retorna un valor verdadero si la pieza puede ser comida por otra sobre el tablero t
	//
	public boolean amenazada (Tablero t) {

		for (int fila=0; fila<8; fila++)
			for (int columna=0; columna<8; columna++)
				if (t.get(fila, columna) != null) {
					Pieza p = t.get(fila, columna);
					if (p.validarMovimiento(t, this.f, this.c))
						return true;
				}
		return false;
	}

	public abstract Pieza copiar ();
	public abstract boolean validarMovimiento (Tablero tablero, int fF, int cF);
	public abstract ArrayList posiblesMovimientos(Tablero t);
}
```

5.11. Reina.java

```
/*
 *$LastChangedDate: 2007-09-18 18:58:03 +0200 (mar, 18 sep 2007) $
 *$LastChangedRevision: 35 $
 * Ligia Tatiana Gonzalez Leyva <calvinahobbes@gmail.com>
 */

import java.util.*;
import java.io.*;

public class Reina extends Pieza {

	// Constructor
	public Reina (int color, int x, int y) {

		super(color, x, y, "imagenes/d");
		//System.out.println("Reina.Reina(): \t\tCreado con color <" + color + "> y
posicion (" + x + ", " + y + ")");
	}

	public String toString () {

		if (color == BLANCO)
			return "D";
		else
			return "d";
	}

	public Pieza copiar () {

		return new Reina(this.color, this.f, this.c);
	}

	public boolean validarMovimiento (Tablero tablero, int fF, int cF) {

		if (fF > 7 || fF < 0)
			return false;
		if (cF > 7 || cF < 0)
			return false;

		Torre t;
		Alfil a;

		//la reina se mueve como una torre o un alfil
		t = new Torre(this.color, this.f, this.c);
```

```
                        a = new Alfil(this.color, this.f, this.c);

                        //si se mueve como una torre o como un alfil, está bien
                        if (t.validarMovimiento(tablero, fF, cF) || a.validarMovimiento(tablero, fF,
cF))
                                return true;
                        return false;
                }

                public ArrayList posiblesMovimientos(Tablero t) {

                        ArrayList a = new ArrayList();

                        for (int df=-7; df<=7; df++)
                                for (int dc=-7; dc<=7; dc++)
                                        if (validarMovimiento(t, this.f+df, this.c+dc))
                                                a.add(new Movimiento(this, this.f+df,
this.c+dc));
                        return a;
                }
}
```

5.12. Rey.java

```
/*
 *$LastChangedDate: 2007-09-18 19:06:57 +0200 (mar, 18 sep 2007) $
 *$LastChangedRevision: 36 $
 * Ligia Tatiana Gonzalez Leyva <calvinahobbes@gmail.com>
 */

import java.util.*;
import java.io.*;

public class Rey extends Pieza {

                // Constructor
                public Rey (int color, int x, int y) {

                        super(color, x, y, "imagenes/r");
                        //System.out.println("Rey.Rey(): \t\tCreado con color <" + color + "> y
posicion (" + x + ", " + y + ")");
                }

                public String toString () {
```

```
                if (color == BLANCO)
                        return "R";
                else
                        return "r";
        }

        public Pieza copiar () {

                return new Rey(this.color, this.f, this.c);
        }

        public boolean validarMovimiento (Tablero tablero, int fF, int cF) {

                if (fF > 7 || fF < 0)
                        return false;
                if (cF > 7 || cF < 0)
                        return false;

                int difFil, difCol;
                Pieza p;

                p = tablero.get(fF, cF);

                difFil = this.f - fF;
                difCol = this.c - cF;

                //solo se mueve una posicion alrededor
                if (Math.abs(difFil) <= 1 && Math.abs(difCol)<= 1) {
                        // solo podemos comer piezas enemigas
                        if (p != null )
                                if (p.getColor() == this.color)
                                        return false;
                        return true;
                }
                else
                        return false;
        }

        public ArrayList posiblesMovimientos(Tablero t) {

                ArrayList a = new ArrayList();

                for (int df=-1; df<=1; df++)
                        for (int dc=-1; dc<=1; dc++)
                                if (validarMovimiento(t, this.f+df, this.c+dc))
                                        a.add(new Movimiento(this, this.f+df,
```

```
this.c+dc));
                        return a;
                }
}
```

5.13. Tablero.java

```
/*
 *$LastChangedDate: 2007-09-18 20:31:00 +0200 (mar, 18 sep 2007) $
 *$LastChangedRevision: 38 $
 * Ligia Tatiana Gonzalez Leyva <calvinahobbes@gmail.com>
 */

import java.util.*;
import java.io.*;
import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;

public class Tablero {

        private JPanel tableroJP;
        private Pieza[][] tablero;
        private Casilla[][] casillas;
        private boolean tablas = false;

        // Constructor
        public Tablero () {

                tablero = new Pieza[8][8];
                casillas = new Casilla[8][8];
                //System.out.println("Tablero.Tablero(): Me acaban de crear");
        }

        // inicializa un tablero como copia de otro (sin interfaz grafica)
        public Tablero (Tablero aCopiar) {

                tablero = new Pieza[8][8];
                casillas = new Casilla[8][8];

                for (int f=0; f<8; f++)
                        for (int c=0; c<8; c++)
                                if (aCopiar.get(f, c) != null)
                                        tablero[f][c] = (aCopiar.get(f,
```

```
c)).copiar();
		}

		// version con GUI
		public Tablero (JFrame f) {

			this();
			Container c = f.getContentPane();
			tableroJP = new JPanel();
			c.add(tableroJP, "Center");
		}

		public void add (int x, int y, Pieza p) {

			tablero[x][y] = p;
		}

		public Pieza get (int f, int c) {

			return tablero[f][c];
		}

		public void mostrar () {

			System.out.println("Tablero.show():");
			for (int f=0; f<8; f++) {
				//System.out.print("F");
				for (int c=0; c<8; c++) {
					if (tablero[f][c] != null)

		System.out.print((tablero[f][c]).toString() + " "); //getClass().getName());
					else
						System.out.print("  ");
				}
				System.out.println();
			}
		}

		// representa la situacion actual del teclado en la pantalla
		//
		public void mostrarGUI () {

			for (int i=0; i < 8; i++) {
				for (int j=0; j < 8; j++) {
					if (tablero[j][i] != null)
```

```
            (casillas[j][i]).setIcon((tablero[j][i]).getImagen());
                                                else
                                                            (casillas[j][i]).setIcon(null);
                                    }
                        }
            }

            // crea el tablero inicialmente vacio
            //
            public void crearTableroGUI (JFrame f) {

            tableroJP.setLayout(new GridLayout(10, 10));
            filaLetras(tableroJP);
            for (int i = 0; i < 8; i++) {
                        tableroJP.add(new BLabel((new Integer(i+1)).toString()));
                        for (int j = 0; j < 8; j++) {
                                                Casilla c;
                                                if (((i + j) % 2) == 0) c = new Casilla(i, j,
Color.white);
                                                else c = new Casilla(i, j, Color.gray);
                                                casillas[i][j] = c;

                        //int color = b.getColor(i, j);
                        //if (color != Board.EMPTY) {
                                    //              int piece = b.getPiece(i, j);
                                    //              square[i][j].setIcon(new
ImageIcon(pieceImage[color][piece]));
                        //}
                        tableroJP.add(casillas[i][j]);
                        }
                        tableroJP.add(new BLabel((new Integer(i+1)).toString()));
            }
            filaLetras(tableroJP);
            }


            // Añade una fila de letras al tablero
            private void filaLetras(JPanel p) {
            p.add(new JPanel());
            for (int i = 0; i < 8; i++)
                        p.add(new BLabel((new Character((char) ('a' + i))).toString()));
            p.add(new JPanel());
            }

            // Genera todos los movimientos validos para el jugador con el color indicado
            public ArrayList generarMovimientos (int color) {
```

```
		ArrayList todas = new ArrayList();
		ArrayList buenas = new ArrayList();
		ArrayList nuevas;
		Pieza actual;

				for (int j=0; j<8; j++)
				for (int i=0; i<8; i++)
				if (tablero[i][j] != null && tablero[i][j].getColor() == color) {
						nuevas = get(i, j).posiblesMovimientos(this);
						todas.addAll(nuevas);
							}
				return todas;
		}

		// Devuelve el valor de un tablero para un jugador con el color indicado. Sus piezas
aportan puntos en funcion
		// de la categoria mientras que las del rival restan igualmente en funcion de su categoria.
		//
		public int valorTablero (int color) {

				// Puntuaciones:
				// Peon: 1
				// Alfil: 3
				// Caballo: 3
				// Torre: 5
				// Reina: 10
				// Rey: 100

				int total = 0;

				for(int i=0; i<8; i++)
						for(int j=0; j<8; j++)
						{
								Pieza pieza = get(i, j);
								if (pieza != null) {
										if (pieza instanceof Peon)

		total+=1*((pieza.getColor()==color)?1:-1);
										else if (pieza instanceof Alfil)

		total+=3*((pieza.getColor()==color)?1:-1);
										else if (pieza instanceof Caballo)

		total+=3*((pieza.getColor()==color)?1:-1);
										else if (pieza instanceof Torre)
```

```
        total+=5*((pieza.getColor()==color)?1:-1);
                                                    else if (pieza instanceof Reina)

        total+=10*((pieza.getColor()==color)?1:-1);
                                                    else if (pieza instanceof Rey)

        total+=100*((pieza.getColor()==color)?1:-1);
                                                }
                }
                return total;
        }

        // Indica que el tablero contiene una partida en situacion de empate
        //
        public void setTablas () {

                tablas = true;
        }

        public boolean getTablas () {

                return tablas;
        }

        class BLabel extends JLabel {
        BLabel(String s) {
                super(s);
                setHorizontalAlignment(CENTER);
                setVerticalAlignment(CENTER);
                }
}

        class Casilla extends JPanel {

                private JLabel l;
                // posicion
                private int f, c;

                public Casilla (int f, int c, Color color) {

                        super();
                        setBackground(color);
                        setPreferredSize(new Dimension(42, 42));
                        this.f = f;
                        this.c = c;
```

```
                        l = new JLabel();
                    l.setPreferredSize(new Dimension(32, 32));
                    add(l);
                        addMouseListener(new CasillaMouseListener());
                    }

                    void setIcon(Icon i) {
                    l.setIcon(i);
                    }

                    class CasillaMouseListener extends MouseAdapter {
                    /*
                        public void mouseEntered(MouseEvent e) {
                        mouseIn = true;
                        repaint();
                    }

                    public void mouseExited(MouseEvent e) {
                        mouseIn = false;
                        repaint();
                    }
                    */
                    public void mouseClicked(MouseEvent e) {
                        //bv.selected(y, x);
                                //System.out.println("Casilla pulsada: <" + f + ", " +
c + ">");
                    }
                    }
            }
}
```

5.14. Torre.java

```
/*
 *$LastChangedDate: 2007-09-18 19:21:56 +0200 (mar, 18 sep 2007) $
 *$LastChangedRevision: 37 $
 * Ligia Tatiana Gonzalez Leyva <calvinahobbes@gmail.com>
 */

import java.util.*;
import java.io.*;

public class Torre extends Pieza {

        // Constructor
```

```
	public Torre (int color, int x, int y) {

		super(color, x, y, "imagenes/t");
		//System.out.println("Torre.Torre(): \t\tCreado con color <" + color + "> y
posicion (" + x + ", " + y + ")");
	}

	public String toString () {

		if (color == BLANCO)
			return "T";
		else
			return "t";
	}

	public Pieza copiar () {

		return new Torre(this.color, this.f, this.c);
	}

	public boolean validarMovimiento (Tablero tablero, int fF, int cF) {

		if (fF > 7 || fF < 0)
			return false;
		if (cF > 7 || cF < 0)
			return false;

		int difFil, difCol, diferencia;
		Pieza p;

		difFil = fF - this.f;
		difCol = cF - this.c;

		p = tablero.get(fF, cF);

		//se debe mover sobre la misma fila o sobre la misma columna
		if ((this.f == fF) || (this.c == cF)) {
			// solo podemos comer una ficha del enemigo
			if (p != null )
				if (p.getColor() == this.color)
					return false;

			// la torre no puede saltar sobre otras fichas
			diferencia = Math.abs(difFil) + Math.abs(difCol);
			for (int i = 1; i < diferencia; i++) {
				p = tablero.get(this.f + i * (difFil / diferencia), this.c
```

```
+ i * (difCol / diferencia));
                                        // la torre trata de saltar
                                        if (p != null )
                                                return false;
                                }
                                return true;
                        }
                        else
                                return false;
                }

                public ArrayList posiblesMovimientos(Tablero t) {

                        ArrayList a = new ArrayList();

                        for (int df=-7; df<=7; df++)
                                if (validarMovimiento(t, this.f+df, this.c))
                                        a.add(new Movimiento(this, this.f+df, this.c));
                        for (int dc=-7; dc<=7; dc++)
                                if (validarMovimiento(t, this.f, this.c+dc))
                                        a.add(new Movimiento(this, this.f, this.c+dc));
                        return a;
                }
}
```