Trabajo Inteligencia Artificial

Jorge Castellano Castellano
David Morales Sáez
2011/2012

Se plantea el desarrollo de una actividad práctica relacionada con la utilización de procedimientos de exploración. Para ello se propone un problema en el escenario del juego conocido como Monkey-Queen.

1 Planteamiento del Problema

En este trabajo de curso, se ha propuesto implementar un conjunto de sistemas de resolución del problema conocido como Monkey-Queen. Para ello, se llevará a cabo un análisis del mismo, estudiando sus distintas variables y propiedades. Además, se diseñará un entorno de juego en el cual los distintos jugadores (ya sean humanos o virtuales) puedan ver las posibles acciones y sus consecuencias.

Para la realización de este trabajo se definirán tanto la estructura con la que se guardarán las distintas partidas como las distintas heurísticas a utilizar, además del uso del sistema y de la representación que se utilizará para poder interactuar con _el. Las heurísticas que se aplicarán son solo una pequeña selección de todo el abanico de posibilidades, cada una con sus ventajas y desventajas.

2 Análisis del Problema

El juego de Monkey Queens es una variación del juego conocido como las Damas, donde cada contendiente tiene, inicialmente, una Reina con una pila de veinte _chas. Las Reinas se pueden mover en cualquier dirección y todo lo que deseen siempre y cuando no pasen por encima de ninguna otra ficha. La Reina se puede mover sin necesidad de comer siempre y cuando tenga una pila de más de dos _chas. Cada vez que esta se mueve (siempre y cuando el movimiento no lleve a comer a una pieza enemiga) genera un Mono y pierde una ficha.

Los Monos tienen los mismos movimientos que las Reinas, pero no generan ninguna pieza. Además, los Monos siempre han de moverse hacia la Reina enemiga, es decir, la nueva posición no puede estar más lejos de la Reina enemiga que en la anterior posición.

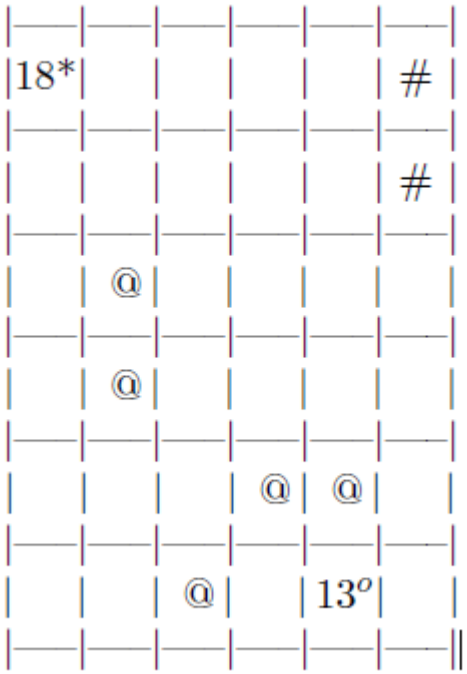
El objetivo del juego es conseguir eliminar a la Reina enemiga, ya sea con un Mono o con la propia Reina, evitando que sea eliminada nuestra Reina.

3 Decisiones del Diseño

El campo de juego será almacenado como una lista formada por las distintas filas, y cada una de estas listas está formada por los distintos elementos que la componen. A nivel de implementación, la representación será la siguiente:

$$\text{Campo de Juego} = [[18^*, 0, 0, 0, 0, \#], Fila2, Fila3, Fila4, Fila5, Fila6]$$

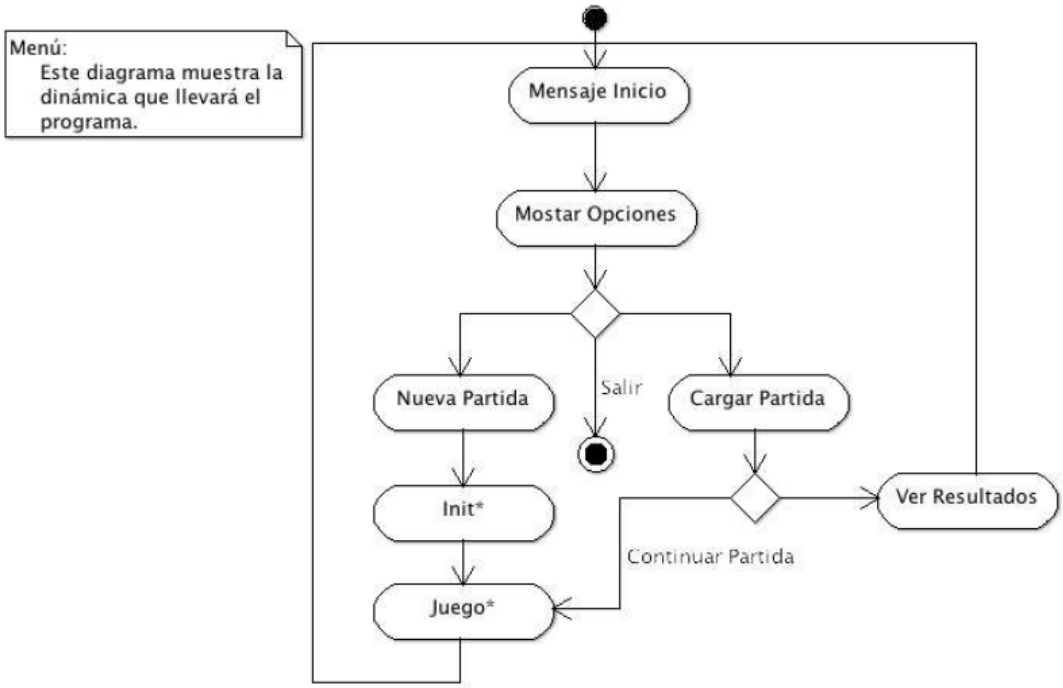
La representación que se mostrará por pantalla será la siguiente:



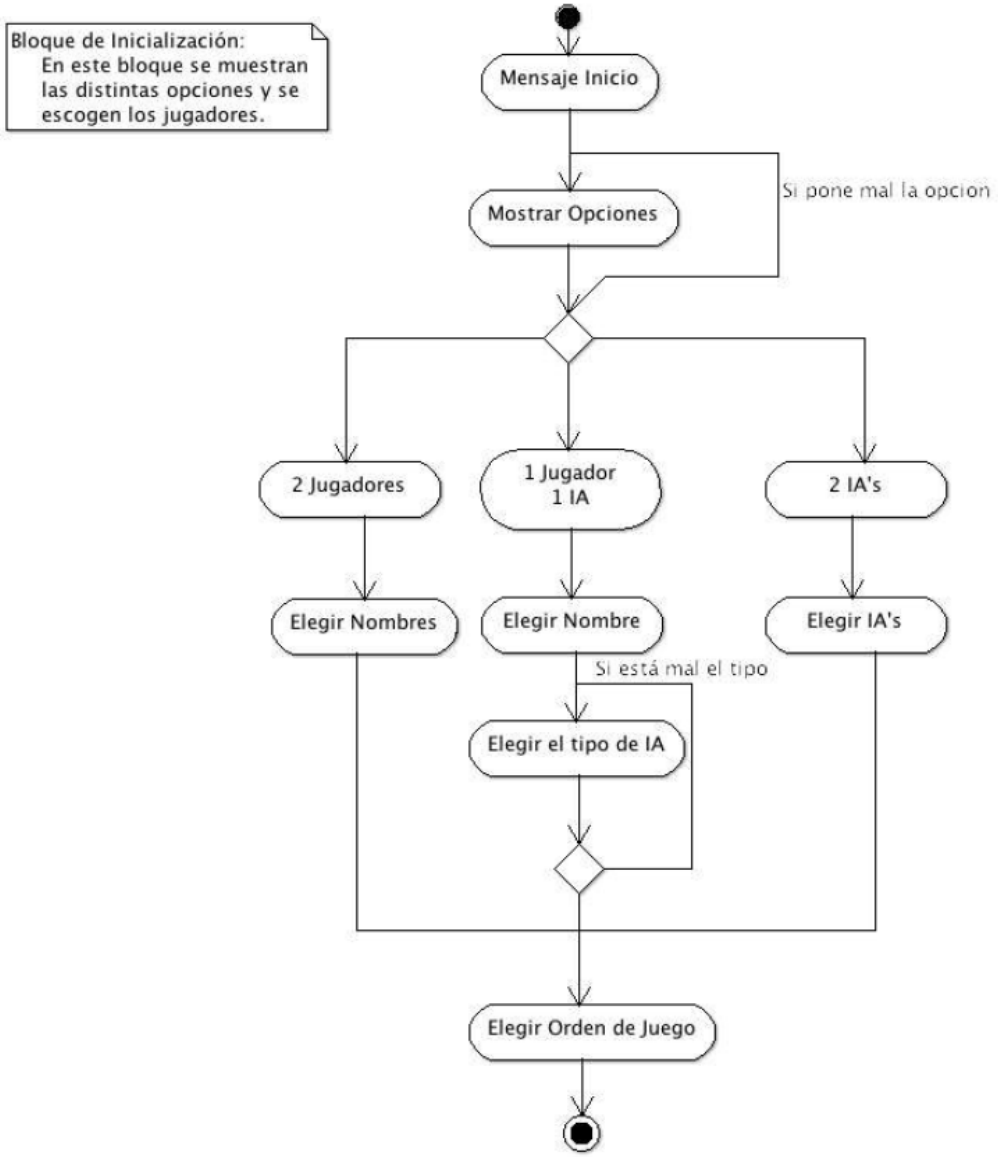
Los distintos jugadores serán representados por los símbolos # y @, y sus reinas serán representadas por N* y No, siendo N el valor de la pila de valores de la reina. El usuario podrá interactuar con el sistema por el teclado, siendo preguntado por los valores necesarios.

Hemos diseñado una serie de diagramas que muestran el flujo del sistema.

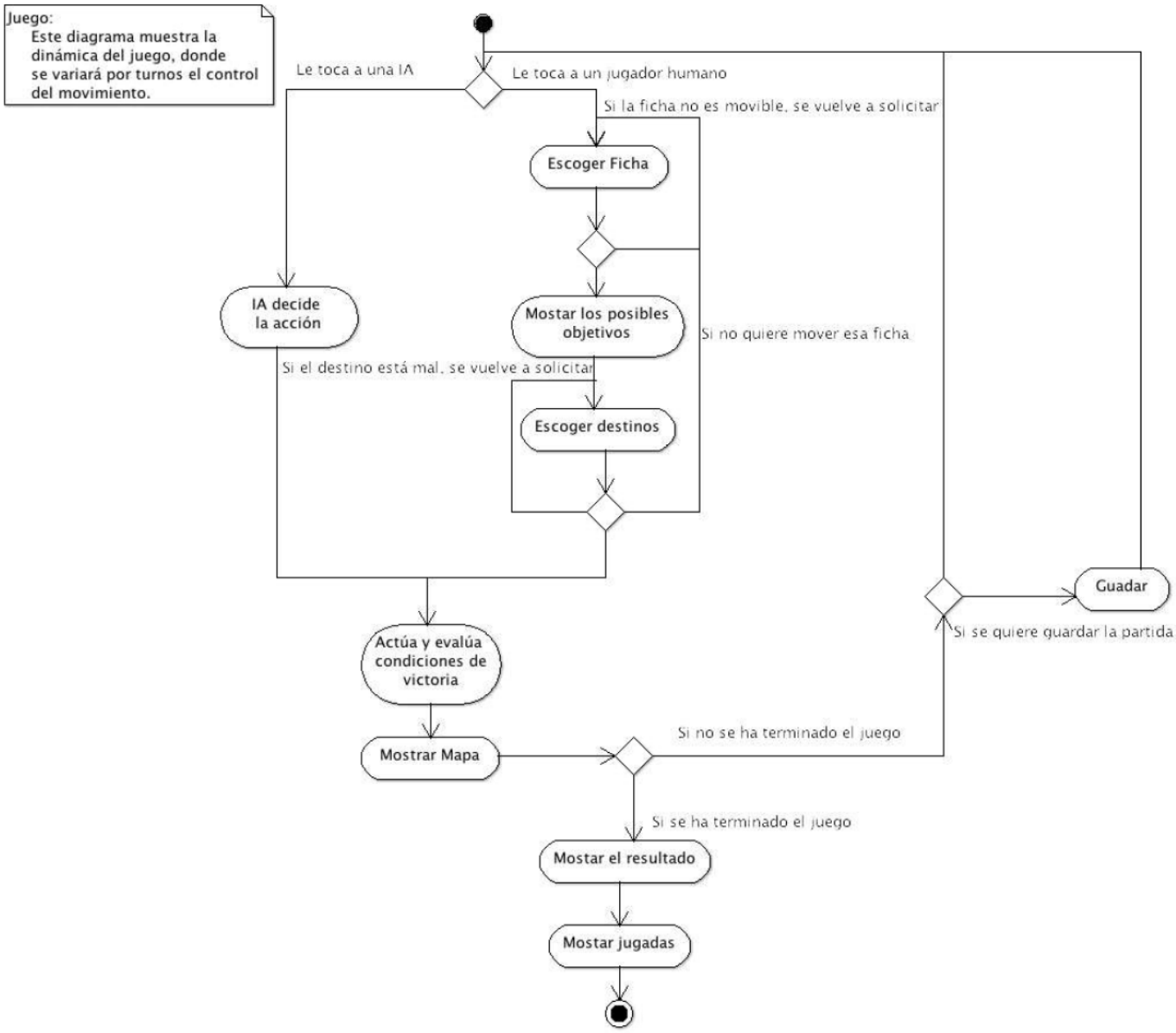
En primer lugar, el diagrama de Actividades del proceso Inicial muestra el comportamiento al inicio del juego, lo que será lo considerado el Menú externo.



El Diagrama del proceso de Inicialización define una nueva partida, estableciendo los distintos parámetros en función de los deseos del jugador.



El Diagrama del proceso de Juego establece el flujo de la partida, variando en función del tipo de jugador y de las actuaciones del mismo.



La representación de la partida será una lista de jugadas ya hechas, con el siguiente formato:

$$[J_{i,1}, J_{i,2}, [J_{i-1,1}, J_{i-1,2}]]$$

siendo las jugadas:

$$J_{i,X} = [Pos\,inicial_x, Pos\,inicial_y, Pos\,final_x, Pos\,final_y, Come|Mueve]$$

A la hora de medir la heurística, las distintas variables que podemos manejar en este problema son las siguientes:

- Número de Monos enemigos.
- Posiciones de Monos enemigos.
- Tamaño de la pila de la Reina enemiga.
- Posición de la Reina enemiga.
- Número de Monos aliados.
- Posiciones de Monos aliados.
- Tamaño de la pila de la Reina aliada.
- Posición de la Reina aliada.

De estas variables podemos obtener las características más importantes:

- Número de Monos enemigos.
- Número de Monos aliados.
- Distancia Media entre los Monos enemigos y la Reina aliada.
- Distancia Media entre los Monos aliados y la Reina enemiga.
- Posibilidad de ataque a la Reina enemiga.
- Posibilidad de ataque a la Reina aliada.

Partiendo de estas características, hemos generado cuatro distintas heurísticas, que consideramos que son las mejores posibles:

1. Rodeamos a la Reina enemiga

$$\frac{Número\ de\ Monos\ Aliados - Número\ de\ Monos\ enemigos + 1}{Distancia\ media\ a\ la\ Reina\ enemiga}$$

Esta heurística busca eliminar al objetivo, obviando la defensa. A medida que los Monos aliados se van acercando a la Reina enemiga, rodeándola y poniéndola en peligro, se van comiendo a los Monos enemigos. Esta heurística permite forzar al enemigo a protegerse, o a reponer los Monos enemigos perdidos, mientras se va acorralando a la Reina enemiga, eliminando posibles movimientos rivales.

El principal problema de esta heurística estriba en la falta de defensa, creando grandes oportunidades al enemigo para acercarse y golpear un golpe fatídico a la Reina aliada.

2. Evitamos cercos a la Reina aliada

$$(Número\ de\ Monos\ Aliados - Número\ de\ Monos\ enemigos + 1)^* \\ Distancia\ Media\ a\ la\ Reina\ aliada$$

En esta heurística evaluamos la diferencia entre los Monos aliados y los enemigos, de modo que procuremos mantener ventaja en el número de monos locales. Esta ventaja la aplicamos a la gestión de la Reina aliada. Nuestra reina, en esta heurística, es sensible a la distancia a la que se encuentran los enemigos, de modo que su actitud será de un carácter defensivo.

Esta heurística se enfrenta a enemigos agresivos con rigidez y adquiere más flexibilidad ante enemigos defensivos de naturaleza similar; aunque se puede derivar en partidas más largas a resolver por desgaste entre los rivales, al no buscar una victoria directa si no la propia supervivencia.

3. Posibilidad de ataque enemigo

$$\frac{\textit{Número de Monos aliados - Número de Monos enemigos + 1}}{\textit{Número de posibles atacantes a la Reina aliada + 1}}$$

Esta heurística busca proteger lo mejor posible a la Reina aliada, buscando todas las zonas de posible riesgo de ataque futuro e intentando minimizarlas. Para ello, exploramos todos los posibles caminos de ataque futuros y buscamos aquella jugada que lo minimice, a la par que intentamos tener una superioridad numérica de Monos frente al enemigo.

La principal falta de esta heurística es que, al intentar proteger a la Reina aliada del enemigo, olvida totalmente la ofensiva, alargando las partidas extremadamente e intentando vencer por desgaste.

4. Posibilidad de ataque aliado

$$\textit{Número de posibles atacantes a la Reina enemiga}^{*}$$
$$\textit{(Número de Monos aliados - Número de Monos enemigos + 1)}$$

Esta heurística busca atacar y eliminar de la forma más aplastante a la Reina enemiga, buscando todos los posibles caminos de ataque.
Para ello, exploramos todos los posibles caminos de ataque futuros y buscamos aquella jugada que lo maximice, a la par que intentamos tener una superioridad numérica de Monos frente al enemigo. Para esta heurística, obviamos totalmente la defensa, permitiendo a los Monos enemigos traspasar nuestras líneas y atacar fácil y fatalmente a la Reina aliada.

4 Implementación. Codificación

Nuestra aplicación está compuesta por 5 módulos básicos y el módulo final de ejecución. Describiremos en estas líneas cada uno de los mismos y sus funcionalidades:

- **MQ_CampoJuego.java**

En esta clase se definirá el campo de juego y el sistema de visionado del mismo por pantalla. Controla la posición de las piezas de cada jugador durante la partida. También evalúa si las piezas se encuentran amenazadas o no por piezas rivales y si tenemos a una ficha enemiga a nuestro alcance.

- **MQ_Heuristica.java**

Clase encargada de manejar cuatro heurísticas definidas en el apartado anterior. Adicionalmente a esta definición se encarga de verificar si la jugada actual es una jugada ganadora o perdedora para, en consecuencia tratarla y actualizar la información necesaria.

- **MQ_Juego.java**

La encargada de controlar el juego en sí, obteniendo las jugadas y actualizando estados. Aquí se comprueban las situaciones entre jugadores y las posibilidades de movimiento legales de cada pieza. La información obtenida se procesa y se selecciona según las heurísticas involucradas en la partida, de modo que se busque la jugada óptima ganadora.

- **MQ_Jugada.java**

Se encarga de guardar una jugada de la partida. Cada jugada es el movimiento de cualquier jugador considerando si este es un movimiento normal o uno de ataque.

- **MQ_Partida.java**

Clase encargada de gestionar los ficheros guardando y cargando los datos desde el archivo que recibe por parámetro. También se encarga de almacenar la sucesión de jugadas de una partida.

- **Trabajo_IA.java**

Modulo principal donde se lanzará el juego y el menú contextual. Se encarga de mostrar y demandar datos al usuario.

El código completo de cada uno de los módulos descritos se encuentra al final del presente documento en el apartado de Apéndices.

5 Manual de Uso

Pará la ejecución de este trabajo será suficiente con abrir un terminal y lanzar el comando “java Trabajo_IA” o importar el proyecto desde Eclipse y ejecutarlo desde el mismo programa de desarrollo.

El archivo relevante de entre los anteriormente descritos a efectos de uso es Trabajo_IA.java.

La interacción con el usuario se desarrolla mediante un menú contextual que demandará al jugador los datos necesarios para establecer el comienzo de una partida y para seguir su desarrollo hasta la finalización, así como para retomar una partida guardada.

La siguiente imagen muestra el menú de entrada de datos y la zona de juego en el primer turno de una partida ejemplo:

```
Bienvenido al Grandioso y Magnífico juego de

                MONKEY-QUEEN



Escoja entre una de las siguientes opciones:

     - MáQUINA vs MÁQUINA (0)
     - JUGADOR vs MÁQUINA (1)
     - JUGADOR vs JUGADOR (2)
     - CARGAR PARTIDA (3)


  Escoja: 1

Por favor, escoja el tamaño del mapa: 9
Por favor, escoja heurística para la máquina: 4
Por favor, escoja la profundidad de la heurística para la máquina: 1
Por favor, escoja el tipo de resolución del árbol de la máquina (MINIMAX = 0, ALFA-BETA = 1): 0
Por favor, escoja la posición X de la reina de la máquina: 8
Por favor, escoja la posición Y de la reina de la máquina: 0
Por favor, escoja la pila de la reina para la máquina: 20
Por favor, escoja la posición X de su reina: 5
Por favor, escoja la posición Y de su reina: 6
Por favor, escoja la pila de su reina: 20
Por favor, indique el nombre del fichero de juego: ejemplo.txt
Por favor, indique quien empieza primero (USTED = 0, MÁQUINA = 1): 1
   | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |20*|
 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
 6 | 0 | 0 | 0 | 0 | 0 |20º| 0 | 0 | 0 |
 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |



   | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
 0 |19*| 0 | 0 | 0 | 0 | 0 | 0 | 0 | # |
 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
 6 | 0 | 0 | 0 | 0 | 0 |20º| 0 | 0 | 0 |
 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |



Por favor, indique la posición X de la ficha a mover:
```

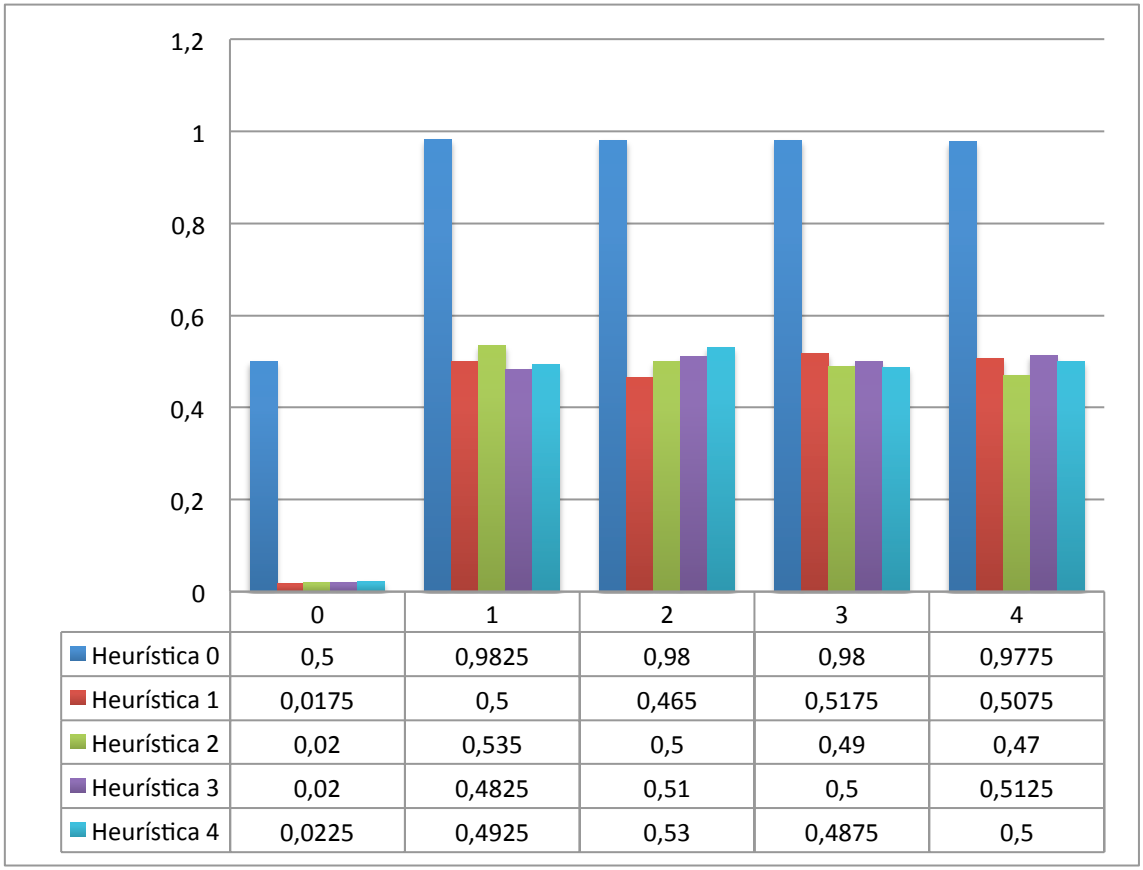
6 Estudio Experimental

Se ha realizado una serie de pruebas para todas las combinaciones posibles de enfrentamiento entre las diferentes heurísticas.

- Heurística 0: Movimiento Aleatorio.
- Heurística 1: Rodeamos a la Reina enemiga.
- Heurística 2: Evitamos cercos a la Reina aliada.
- Heurística 3: Posibilidad de ataque enemigo.
- Heurística 4: Posibilidad de ataque aliado.

También se han probado las mismas heurísticas a diferentes niveles de profundidad, sector limitado por la capacidad de cómputo del sistema.

El siguiente gráfico muestra los resultados en porcentaje de victorias obtenidas entre dos usuarios máquina con la misma o diferentes heurísticas y con las variaciones de profundidad reseñadas tras 1000 enfrentamientos en cada caso:



	0	1	2	3	4
Heurística 0	0,5	0,9825	0,98	0,98	0,9775
Heurística 1	0,0175	0,5	0,465	0,5175	0,5075
Heurística 2	0,02	0,535	0,5	0,49	0,47
Heurística 3	0,02	0,4825	0,51	0,5	0,5125
Heurística 4	0,0225	0,4925	0,53	0,4875	0,5

7 Conclusiones

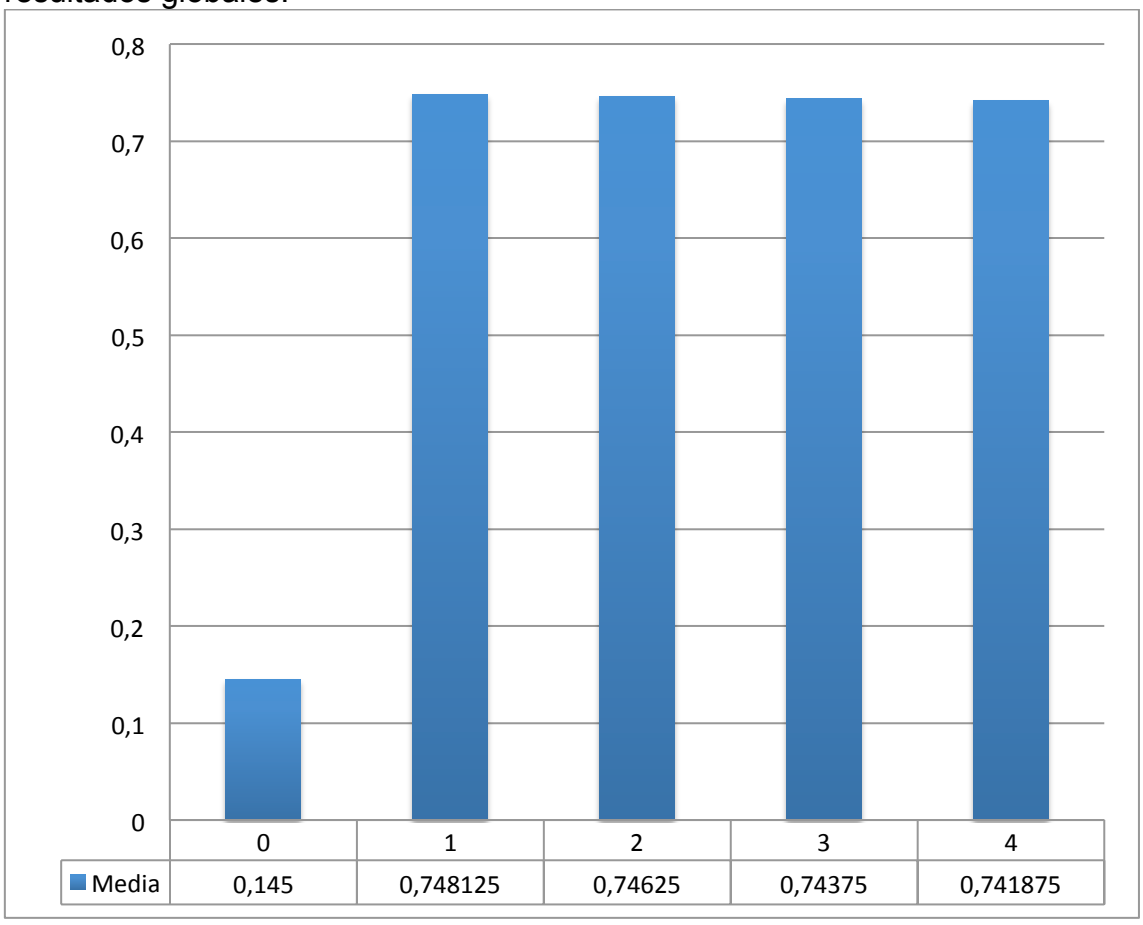
Analizando los resultados del apartado anterior, podemos apreciar como la Heurística 0 obtiene unos resultados pésimos contra el resto de oponentes, debido a que se basa en la aleatoriedad y ni siquiera evita los ataques suicidas.

La Heurística 1 se presenta como la vencedora, al obtener unas tasas de victoria superiores contra los oponentes. La estrategia ofensiva de desgaste planteada al rodear a la Reina enemiga resulta muy eficaz contra el resto de Heurísticas.

En el caso de la Heurística 2 que presenta un corte defensivo vemos como es eficaz contra las demás, pero como cede ante la Heurística 1, pues cae en su trampa y al defender nuestra reina hacemos precisamente lo que busca el rival, que es rodearnos.

Las Heurísticas 3 y 4 al basarse en posibilidades no resultan tan elaboradas como la 1 y la 2 y sucumben ante ellas. Se confirma como en su enfrentamiento particular la 3, que se defiende, logra vencer a la 4 que ataca sin un rumbo tan claro.

En el siguiente gráfico vemos la media de victorias con respecto a los resultados anteriores, por lo que nos sirve como valor final para evaluar los resultados globales:



La clasificación general de las heurísticas y sus características de acción quedaría entonces de la siguiente manera:

1. Heurística 1: Ofensiva focalizada.
2. Heurística 2: Defensiva focalizada.
3. Heurística 3: Defensiva simple.
4. Heurística 4: Ofensiva simple.
5. Heurística 0: Aleatorio

Destacar los resultados de la heurística aleatoria ya que es la menos inteligente, es decir, no evita ni suicidios, y la más variante de las descritas y la contundencia de las derrotas sufridas por la heurística 4 en su afán ofensivo no controlado.

Tenemos por tanto unos resultados que se corresponden con las previsiones previas y con la lógica general de las estrategias clásicas de ataque y defensa.

8 Apéndices

Trabajo_IA.java

```
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Random;
import java.util.Scanner;
import monkeyqueen.*;

public class Trabajo_IA {

	public static MQ_Partida partida;


	public static void main(String[] args)
	{
		while(true)
		{
			partida = new MQ_Partida();
			Scanner sc = new Scanner(System.in);
			System.out.print("\nBienvenido al Grandioso y Magnífico juego
de \n");
			System.out.print("\n                MONKEY-QUEEN
\n\n\n");
			System.out.print("\nEscoja entre una de las siguientes
opciones: \n");
			System.out.print("\n     - MáQUINA vs MÁQUINA (0)");
			System.out.print("\n     - JUGADOR vs MÁQUINA (1)");
			System.out.print("\n     - JUGADOR vs JUGADOR (2)");
			System.out.print("\n     - CARGAR PARTIDA (3)");
			System.out.print("\n\n\n  Escoja: ");
			int opc = sc.nextInt();
			switch(opc)
			{
			case 0:
				Modo_2IA(false);
				break;
			case 1:
				Modo_1IA(false);
				break;
			case 2:
				Modo_2Jug(false);
				break;
			case 3:
				CargarPartida();
				break;
			}
		}
	}

	public static void CargarPartida()
	{
		String ruta = "";
		Scanner sc = new Scanner(System.in);
		System.out.print("\nPor favor, indique la ruta de la partida: ");
		ruta = sc.next();
		partida.Carga_Partida(ruta);
		System.out.print("\nEscoja entre una de las siguientes opciones: \n");
		System.out.print("\n     - MçQUINA vs MçQUINA (0)");
		System.out.print("\n     - JUGADOR vs MçQUINA (1)");
```

```
		System.out.print("\n     - JUGADOR vs JUGADOR (2)");
		System.out.print("\n     - VER JUEGO (3)");
		System.out.print("\n\n\n  Escoja: ");
		int opc = sc.nextInt();
		switch(opc)
		{
		case 0:
			Modo_2IA(true);
			break;
		case 1:
			Modo_1IA(true);
			break;
		case 2:
			Modo_2Jug(true);
			break;
		case 3:
			Ver_Historial();
			break;
		}
	}

	public static void Ver_Historial()
	{
		int ia1, ia2, dim, tam1, tam2, posx1, posx2, posy1, posy2, prof1,
prof2, resol1, resol2;
		int valores[] = new int[7];
		valores = partida.ValoresIniciales();
		dim = valores[0];
		tam1 = valores[1];
		tam2 = valores[2];
		posx1 = valores[3];
		posy1 = valores[4];
		posx2 = valores[5];
		posy2 = valores[6];
		ia1 = valores[7];
		prof1 = valores[8];
		resol1 = valores[9];
		ia2 = valores[10];
		prof2 = valores[11];
		resol2 = valores[12];
		MQ_Juego juego = new MQ_Juego(dim, ia1, ia2, tam1, tam2, prof1, prof2,
resol1, resol2, posx1, posy1, posx2, posy2, 0);
		for(int i=0; i<partida.NumeroJugadas(); i++)
		{
			juego.Juega_Jugada(partida.ObtenerJugada(i));
			juego.Imprime();
		}
	}

	public static void Modo_2IA(boolean cargado)
	{
		int ia1, ia2, dim, tam1, tam2, posx1, posx2, posy1, posy2, prof1,
prof2, resol1, resol2;
		Scanner sc = new Scanner(System.in);
		String ruta;
		if(!cargado)
		{
			System.out.print("\nPor favor, El tamaño del mapa: ");
			dim = sc.nextInt();
			while(true)
			{
				System.out.print("Por favor, escoja la posicion X de la
reina del jugador 1: ");
				posy1 = sc.nextInt();
				System.out.print("Por favor, escoja la posicion Y de la
reina del jugador 1: ");
```

```
                        posx1 = sc.nextInt();
                        if((posy1<0)||(posy1>=dim)||(posx1<0)||(posx1>=dim))
                        {
                                System.out.print("La posicion indicada esta fuera
de rango\n");
                                continue;
                        }
                        break;
                }
                System.out.print("Por favor, escoja la pila de la reina para el
jugador 1: ");
                tam1 = sc.nextInt();
                while(true)
                {
                        System.out.print("Por favor, escoja la posicion X de la
reina del jugador 2: ");
                        posy2 = sc.nextInt();
                        System.out.print("Por favor, escoja la posicion Y de la
reina del jugador 2: ");
                        posx2 = sc.nextInt();
                        if((posy2<0)||(posy2>=dim)||(posx2<0)||(posx2>=dim))
                        {
                                System.out.print("La posicion indicada esta fuera
de rango\n");
                                continue;
                        }
                        break;
                }
                System.out.print("Por favor, escoja la pila de la reina para el
jugador 2: ");
                tam2 = sc.nextInt();
                System.out.print("Por favor, escoja heurística para el jugador
1: ");
                ia1 = sc.nextInt();
                System.out.print("Por favor, escoja la profundidad de la
heurística para el jugador 1: ");
                prof1 = sc.nextInt();
                System.out.print("Por favor, escoja el tipo de resolución del
‡rbol del jugador 1 (MINIMAX = 0, ALFA-BETA = 1): ");
                resol1 = sc.nextInt();
                System.out.print("Por favor, escoja heurística para el jugador
2: ");
                ia2 = sc.nextInt();
                System.out.print("Por favor, escoja la profundidad de la
heurística para el jugador 2: ");
                prof2 = sc.nextInt();
                System.out.print("Por favor, escoja el tipo de resolución del
árbol del jugador 2 (MINIMAX = 0, ALFA-BETA = 1): ");
                resol2 = sc.nextInt();
                System.out.print("Por favor, indique el nombre del fichero de
juego: ");
                ruta = sc.next();
                partida.Set_Ruta(ruta);

                int valores[] = new int[13];
                valores[0] = dim;
                valores[1] = tam1;
                valores[2] = tam2;
                valores[3] = posx1;
                valores[4] = posy1;
                valores[5] = posx2;
                valores[6] = posy2;
                valores[7] = ia1;
                valores[8] = prof1;
                valores[9] = resol1;
                valores[10] = ia2;
```

```
                        valores[11] = prof2;
                        valores[12] = resol2;
                        partida.SetValoresIniciales(valores);
                        partida.Salva_Partida();
                }
                else
                {
                        int valores[] = new int[7];
                        valores = partida.ValoresIniciales();
                        dim = valores[0];
                        tam1 = valores[1];
                        tam2 = valores[2];
                        posx1 = valores[3];
                        posy1 = valores[4];
                        posx2 = valores[5];
                        posy2 = valores[6];
                        ia1 = valores[7];
                        prof1 = valores[8];
                        resol1 = valores[9];
                        ia2 = valores[10];
                        prof2 = valores[11];
                        resol2 = valores[12];
                        ruta = partida.Get_ruta();
                }
                MQ_Juego juego = new MQ_Juego(dim, ia1, ia2, tam1, tam2, prof1, prof2,
resol1, resol2, posx1, posy1, posx2, posy2, 0);
                if(cargado)
                {
                        for(int i=0; i<partida.NumeroJugadas(); i++)
                                juego.Juega_Jugada(partida.ObtenerJugada(i));
                }
                juego.Imprime();
                while(true)
                {
                        juego.Juega(0, partida);
                        juego.Imprime();
                        int fin = juego.CompruebaFin();
                        if(fin==1)
                        {
                                System.out.print("El Jugador 1 ha ganado");
                                partida.Salva_Partida();
                                return;
                        }
                        juego.Juega(1, partida);
                        juego.Imprime();
                        fin = juego.CompruebaFin();
                        if(fin==-1)
                        {
                                System.out.print("El Jugador 2 ha ganado");
                                partida.Salva_Partida();
                                return;
                        }
                        partida.Salva_Partida();
                }
        }

        public static void Modo_1IA(boolean cargado)
        {
                int ia1, dim, tam1, tam2, posx1, posx2, posy1, posy2, prof1, resol1;
                MQ_Juego juego;
                String ruta;
                Scanner sc = new Scanner(System.in);
                if(!cargado)
                {
                        System.out.print("\nPor favor, escoja el tamaño del mapa: ");
                        dim = sc.nextInt();
```

```
                        System.out.print("Por favor, escoja heurística para la máquina:
");
                        ia1 = sc.nextInt();
                        System.out.print("Por favor, escoja la profundidad de la
heurística para la máquina: ");
                        prof1 = sc.nextInt();
                        System.out.print("Por favor, escoja el tipo de resolución del
árbol de la máquina (MINIMAX = 0, ALFA-BETA = 1): ");
                        resol1 = sc.nextInt();
                        while(true)
                        {
                                System.out.print("Por favor, escoja la posición X de la
reina de la máquina: ");
                                posy1 = sc.nextInt();
                                System.out.print("Por favor, escoja la posición Y de la
reina de la máquina: ");
                                posx1 = sc.nextInt();
                                if((posy1<0)||(posy1>=dim)||(posx1<0)||(posx1>=dim))
                                {
                                        System.out.print("La posición indicada está fuera
de rango\n");
                                        continue;
                                }
                                break;
                        }
                        System.out.print("Por favor, escoja la pila de la reina para la
máquina: ");
                        tam1 = sc.nextInt();
                        while(true)
                        {
                                System.out.print("Por favor, escoja la posición X de su
reina: ");
                                posy2 = sc.nextInt();
                                System.out.print("Por favor, escoja la posición Y de su
reina: ");
                                posx2 = sc.nextInt();
                                if((posy2<0)||(posy2>=dim)||(posx2<0)||(posx2>=dim))
                                {
                                        System.out.print("La posición indicada está fuera
de rango\n");
                                        continue;
                                }
                                break;
                        }
                        System.out.print("Por favor, escoja la pila de su reina: ");
                        tam2 = sc.nextInt();
                        System.out.print("Por favor, indique el nombre del fichero de
juego: ");
                        ruta = sc.next();
                        partida.Set_Ruta(ruta);

                        int valores[] = new int[13];
                        valores[0] = dim;
                        valores[1] = tam1;
                        valores[2] = tam2;
                        valores[3] = posx1;
                        valores[4] = posy1;
                        valores[5] = posx2;
                        valores[6] = posy2;
                        valores[7] = ia1;
                        valores[8] = prof1;
                        valores[9] = resol1;
                        valores[10] = 0;
                        valores[11] = 1;
                        valores[12] = 0;
                        partida.SetValoresIniciales(valores);
```

```
                        partida.Salva_Partida();
                }
                else
                {
                        int valores[] = new int[7];
                        valores = partida.ValoresIniciales();
                        dim = valores[0];
                        tam1 = valores[1];
                        tam2 = valores[2];
                        posx1 = valores[3];
                        posy1 = valores[4];
                        posx2 = valores[5];
                        posy2 = valores[6];
                        ia1 = valores[7];
                        prof1 = valores[8];
                        resol1 = valores[9];
                        ruta = partida.Get_ruta();
                }
                juego = new MQ_Juego(dim, ia1, 0, tam1, tam2, prof1, 0, resol1, 0,
posx1, posy1, posx2, posy2, 0);
                int opc;
                if(cargado)
                {
                        for(int i=0; i<partida.NumeroJugadas(); i++)
                                juego.Juega_Jugada(partida.ObtenerJugada(i));
                        if(partida.NumeroJugadas()==0)
                        {

                                System.out.print("Por favor, indique quien empieza
primero (USTED = 0, MÁQUINA = 1): ");
                                opc = sc.nextInt();
                        }
                        else
                        {
                                opc = partida.NumeroJugadas()%2;
                        }
                }
                else
                {
                        System.out.print("Por favor, indique quien empieza primero
(USTED = 0, MÁQUINA = 1): ");
                        opc = sc.nextInt();
                }
                if(opc==0)
                {
                        juego.Cambia_Jugador();
                        juego.Imprime();
                        while(true)
                        {
                                System.out.print("\nPor favor, indique la posición X de
la ficha a mover: ");
                                posx1 = sc.nextInt();
                                System.out.print("\nPor favor, indique la posición Y de
la ficha a mover: ");
                                posy1 = sc.nextInt();
                                if(juego.ObtenerFichaJugador(posx1, posy1)!=1)
                                {
                                        System.out.print("\nLa ficha escogida no puede
ser movida");
                                        continue;
                                }
                                System.out.print("\nPor favor, indique la posición X de
la posición a mover: ");
                                posy2 = sc.nextInt();
                                System.out.print("\nPor favor, indique la posición Y de
la posición a mover: ");
```

```
                        posx2 = sc.nextInt();
                        int ficha = juego.ObtenerFichaJugador(posx2, posy2);
                        if(ficha==1)
                        {
                                System.out.print("\nLa ficha no puede ser movida
a la posición escogida");
                                continue;
                        }
                        MQ_Jugada jugada;
                        if(ficha==1)
                        {
                                jugada = new MQ_Jugada(posx1, posx2, posy1,
posy2, 0);
                                if(!juego.Comprueba_Jugada(jugada))
                                {
                                        System.out.print("\nLa ficha no puede ser
movida a la posición escogida");
                                        continue;
                                }
                                juego.Juega_Jugada(jugada);
                                partida.Inserta_Jugada(jugada);
                        }
                        if(ficha==-1)
                        {
                                jugada = new MQ_Jugada(posx1, posx2, posy1,
posy2, 1);
                                if(!juego.Comprueba_Jugada(jugada))
                                {
                                        System.out.print("\nLa ficha no puede ser
movida a la posición escogida");
                                        continue;
                                }
                                juego.Juega_Jugada(jugada);
                                partida.Inserta_Jugada(jugada);
                        }
                        juego.Imprime();
                        int fin = juego.CompruebaFin();
                        if(fin==-1)
                        {
                                System.out.print("Felicidades, has ganado!");
                                partida.Salva_Partida();
                                return;
                        }
                        juego.Juega(0, partida);
                        juego.Imprime();
                        fin = juego.CompruebaFin();
                        if(fin==1)
                        {
                                System.out.print("Ohhh, has perdido ... ");
                                partida.Salva_Partida();
                                return;
                        }
                        partida.Salva_Partida();
                        ruta = partida.Get_ruta();
                }
        }
        else
        {
                juego.Imprime();
                while(true)
                {
                        juego.Juega(0, partida);
                        juego.Imprime();
                        int fin = juego.CompruebaFin();
                        if(fin==-1)
                        {
```

```
                                System.out.print("La máquina ha ganado");
                                partida.Salva_Partida();
                                return;
                        }
                        MQ_Jugada jugada;
                        while(true)
                        {
                                System.out.print("\nPor favor, indique la
posición X de la ficha a mover: ");
                                posy1 = sc.nextInt();
                                System.out.print("\nPor favor, indique la
posición Y de la ficha a mover: ");
                                posx1 = sc.nextInt();
                                if(juego.ObtenerFichaJugador(posx1, posy1)!=1)
                                {
                                        System.out.print("\nLa ficha escogida no
puede ser movida");
                                        continue;
                                }
                                System.out.print("\nPor favor, indique la
posición X de la posición a mover: ");
                                posy2 = sc.nextInt();
                                System.out.print("\nPor favor, indique la
posición Y de la posición a mover: ");
                                posx2 = sc.nextInt();
                                int ficha = juego.ObtenerFichaJugador(posx2,
posy2);
                                if(ficha==1)
                                {
                                        System.out.print("\n1.La ficha no puede
ser movida a la posición escogida");
                                        continue;
                                }
                                if(ficha==0)
                                {
                                        jugada = new MQ_Jugada(posx1, posx2,
posy1, posy2, 0);
                                        if(!juego.Comprueba_Jugada(jugada))
                                        {
                                                System.out.print("\n0.La ficha no
puede ser movida a la posición escogida");
                                                continue;
                                        }
                                        juego.Juega_Jugada(jugada);
                                        partida.Inserta_Jugada(jugada);
                                }
                                if(ficha==-1)
                                {
                                        jugada = new MQ_Jugada(posx1, posx2,
posy1, posy2, 1);
                                        if(!juego.Comprueba_Jugada(jugada))
                                        {
                                                System.out.print("\n-1.La ficha no
puede ser movida a la posición escogida");
                                                continue;
                                        }
                                        juego.Juega_Jugada(jugada);
                                        partida.Inserta_Jugada(jugada);
                                }
                                break;
                        }
                        juego.Imprime();
                        fin = juego.CompruebaFin();
                        if(fin==1)
                        {
                                System.out.print("Has ganado!");
```

```
                                        partida.Salva_Partida();
                                        return;
                                }
                        }
                }
        }

        public static void Modo_2Jug(boolean cargado)
        {
                int dim, tam1, tam2, posx1, posx2, posy1, posy2;
                String ruta;
                Scanner sc = new Scanner(System.in);
                if(!cargado)
                {
                        System.out.print("\nPor favor, escoja el tamaño del mapa: ");
                        dim = sc.nextInt();
                        while(true)
                        {
                                System.out.print("Por favor, escoja la posición X de la
reina del jugador 1: ");
                                posy1 = sc.nextInt();
                                System.out.print("Por favor, escoja la posición Y de la
reina del jugador 1: ");
                                posx1 = sc.nextInt();
                                if((posy1<0)||(posy1>=dim)||(posx1<0)||(posx1>=dim))
                                {
                                        System.out.print("La posición indicada está fuera
de rango\n");
                                        continue;
                                }
                                break;
                        }
                        System.out.print("Por favor, escoja la pila de la reina del
jugador 1: ");
                        tam1 = sc.nextInt();
                        while(true)
                        {
                                System.out.print("Por favor, escoja la posición X de la
reina del jugador 2: ");
                                posy2 = sc.nextInt();
                                System.out.print("Por favor, escoja la posición Y de la
reina del jugador 2: ");
                                posx2 = sc.nextInt();
                                if((posy2<0)||(posy2>=dim)||(posx2<0)||(posx2>=dim))
                                {
                                        System.out.print("La posición indicada está fuera
de rango\n");
                                        continue;
                                }
                                break;
                        }
                        System.out.print("Por favor, escoja la pila de la reina del
jugador 2: ");
                        tam2 = sc.nextInt();
                        System.out.print("Por favor, indique el nombre del fichero de
juego: ");
                        ruta = sc.next();
                        partida.Set_Ruta(ruta);

                        int valores[] = new int[13];
                        valores[0] = dim;
                        valores[1] = tam1;
                        valores[2] = tam2;
                        valores[3] = posx1;
                        valores[4] = posy1;
                        valores[5] = posx2;
```

```
                        valores[6] = posy2;
                        valores[7] = 0;
                        valores[8] = 1;
                        valores[9] = 0;
                        valores[10] = 0;
                        valores[11] = 1;
                        valores[12] = 0;
                        partida.SetValoresIniciales(valores);
                        partida.Salva_Partida();
                }
                else
                {
                        int valores[] = new int[7];
                        valores = partida.ValoresIniciales();
                        dim = valores[0];
                        tam1 = valores[1];
                        tam2 = valores[2];
                        posx1 = valores[3];
                        posy1 = valores[4];
                        posx2 = valores[5];
                        posy2 = valores[6];
                        ruta = partida.Get_ruta();
                }
                MQ_Juego juego = new MQ_Juego(dim, 0, 0, tam1, tam2, 0, 0, 0, 0,
posx1, posy1, posx2, posy2, 0);
                juego.Imprime();
                while(true)
                {
                        System.out.print("\nJugador 1: Por favor, indique la posición X
de la ficha a mover: ");
                        posy1 = sc.nextInt();
                        System.out.print("\nJugador 1: Por favor, indique la posición Y
de la ficha a mover: ");
                        posx1 = sc.nextInt();
                        if(juego.ObtenerFichaJugador(posx1, posy1)!=0)
                        {
                                System.out.print("\nJugador 1: La ficha escogida no
puede ser movida");
                                continue;
                        }
                        System.out.print("\nJugador 1: Por favor, indique la posición X
de la posición a mover: ");
                        posy2 = sc.nextInt();
                        System.out.print("\nJugador 1: Por favor, indique la posición Y
de la posición a mover: ");
                        posx2 = sc.nextInt();
                        int ficha = juego.ObtenerFichaJugador(posx2, posy2);
                        if(ficha==0)
                        {
                                System.out.print("\nJugador 1: La ficha no puede ser
movida a la posición escogida");
                                continue;
                        }
                        MQ_Jugada jugada;
                        if(ficha==1)
                        {
                                jugada = new MQ_Jugada(posx1, posx2, posy1, posy2, 0);
                                if(!juego.Comprueba_Jugada(jugada))
                                {
                                        System.out.print("\nJugador 1: La ficha no puede
ser movida a la posición escogida");
                                        continue;
                                }
                                juego.Juega_Jugada(jugada);
                                partida.Inserta_Jugada(jugada);
                        }
```

```
                        if(ficha==-1)
                        {
                                jugada = new MQ_Jugada(posx1, posx2, posy1, posy2, 1);
                                if(!juego.Comprueba_Jugada(jugada))
                                {
                                        System.out.print("\nJugador 1: La ficha no puede
ser movida a la posición escogida");
                                        continue;
                                }
                                juego.Juega_Jugada(jugada);
                                partida.Inserta_Jugada(jugada);
                        }
                        if(ficha==-10)
                        {
                                System.out.print("\nLa posición indicada está fuera de
rango.");
                                continue;
                        }
                        juego.Imprime();
                        int fin = juego.CompruebaFin();
                        if(fin==1)
                        {
                                System.out.print("Ha ganado el jugador 1");
                                partida.Salva_Partida();
                                return;
                        }
                        while(true)
                        {
                                System.out.print("\nJugador 2: Por favor, indique la
posición X de la ficha a mover: ");
                                posy1 = sc.nextInt();
                                System.out.print("\nJugador 2: Por favor, indique la
posición Y de la ficha a mover: ");
                                posx1 = sc.nextInt();
                                if(juego.ObtenerFichaJugador(posx1, posy1)!=1)
                                {
                                        System.out.print("\nJugador 2: La ficha escogida
no puede ser movida");
                                        continue;
                                }
                                System.out.print("\nJugador 2: Por favor, indique la
posición X de la posición a mover: ");
                                posy2 = sc.nextInt();
                                System.out.print("\nJugador 2: Por favor, indique la
posición Y de la posición a mover: ");
                                posx2 = sc.nextInt();
                                ficha = juego.ObtenerFichaJugador(posx2, posy2);
                                if(ficha==1)
                                {
                                        System.out.print("\n0. Jugador 2: La ficha no
puede ser movida a la posición escogida");
                                        continue;
                                }
                                MQ_Jugada jugada2;
                                if(ficha==-1)
                                {
                                        jugada2 = new MQ_Jugada(posx1, posx2, posy1,
posy2, 1);
                                        if(!juego.Comprueba_Jugada(jugada2))
                                        {
                                                System.out.print("\n1. Jugador 2: La ficha
no puede ser movida a la posición escogida");
                                                continue;
                                        }
                                        juego.Juega_Jugada(jugada2);
                                        partida.Inserta_Jugada(jugada2);
```

```
                        }
                        if(ficha==0)
                        {
                                jugada2 = new MQ_Jugada(posx1, posx2, posy1,
posy2, 0);
                                if(!juego.Comprueba_Jugada(jugada2))
                                {
                                        System.out.print("\n-1. Jugador 2: La
ficha no puede ser movida a la posición escogida");
                                        continue;
                                }
                                juego.Juega_Jugada(jugada2);
                                partida.Inserta_Jugada(jugada2);
                        }
                        juego.Imprime();
                        fin = juego.CompruebaFin();
                        if(fin==-1)
                        {
                                System.out.print("Ha ganado el jugador 2");
                                partida.Salva_Partida();
                                return;
                        }
                        break;
                }
                partida.Salva_Partida();
        }
    }
}
```

MQ_CampoJuego.java

```
package monkeyqueen;

import java.lang.Math;

	/* * * * * * * * * * * * * * * * * * * * * * * * * * * *
	 * Esta clase define el campo de juego y establece   *
	 * el sistema de impresi—n por pantalla.             *
	 * * * * * * * * * * * * * * * * * * * * * * * * * * * */

public class MQ_CampoJuego
{
	/* * * * * * * * * * * * * * * * * * * * * * * * * * * *
	 * Las variables del campo de juego son:                *
	 *            - Tam: Tama–o de la matriz (tam*tam)            *
	 *            - pos(x,y)Reina1: Posici—n de la Reina 1        *
	 *            - pos(x,y)Reina2: Posici—n de la Reina 2        *
	 *            - mapa: representaci—n matricial del campo      *
	 *                        de juego. Las piezas del jugador 1      *
	 *                        se representar‡n en positivo y las      *
	 *                        del jugador 2 en negativo. Las reinas   *
	 *                        se representar‡n con la cantidad de     *
	 *                        fichas que disponen.                    *
	 * * * * * * * * * * * * * * * * * * * * * * * * * * * */
	private int tam;
	private int posxReina1;
	private int posyReina1;
	private int posxReina2;
	private int posyReina2;
	private int mapa[][];
	private int jugador;
	public boolean tablas;

	public MQ_CampoJuego(int tamano, int xR1, int yR1, int tam1, int xR2, int yR2,
int tam2)
	{
		tam = tamano;
		mapa = new int[tam][tam];
		for(int i=0; i<tam; i++)
			for(int j=0; j<tam; j++)
				mapa[i][j] = 0;
		posxReina1 = xR1;
		posyReina1 = yR1;
		posxReina2 = xR2;
		posyReina2 = yR2;
		mapa[posxReina1][posyReina1] = tam1;
		mapa[posxReina2][posyReina2] = tam2*-1;
		jugador = 0;
		tablas = false;
	}

	public MQ_CampoJuego(int[][] map, int tamano)
	{
		tam = tamano;
		mapa = new int[tam][tam];
		for(int i=0; i<tam; i++)
		{
			for(int j=0; j<tam; j++)
			{
				mapa[i][j] = map[i][j];
				if(Math.abs(mapa[i][j])>1)
				{
					if(mapa[i][j]<0)
					{
```

```
                                        posxReina2 = i;
                                        posyReina2 = j;
                                }
                                else
                                {
                                        posxReina1 = i;
                                        posyReina1 = j;
                                }
                        }
                }
        }
        tablas = false;
}

public MQ_CampoJuego(MQ_CampoJuego campo)
{
        tam = campo.Devuelve_Dimension();
        posxReina1 = campo.Devuelve_PosXReina1();
        posyReina1 = campo.Devuelve_PosYReina1();
        posxReina2 = campo.Devuelve_PosXReina2();
        posyReina2 = campo.Devuelve_PosYReina2();
        mapa = new int[tam][tam];
        for(int i=0; i<tam; i++)
                for(int j=0; j<tam; j++)
                        mapa[i][j] = campo.Devuelve_Valor(i, j);
        jugador = campo.Devuelve_Jugador();
}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 * Esta funci—n comprueba si la ficha enemiga objetivo     *
 * est‡ a tiro, es decir, si ambas fichas est‡n alineadas  *
 * y si no hay ninguna otra ficha que obstaculice el       *
 * ataque     .                                                                *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

public boolean Conectados(int x1, int y1, int x2, int y2)
{
        if(x1 == x2)
        {
                if(y1==y2+1||y1==y2-1)
                        return true;
                int alpha;
                if(y1<y2)
                        alpha = 1;
                else
                        alpha = -1;
                for(int i=alpha;  Math.abs(i)<Math.abs(y2-y1); i+= alpha)
                {
                        if(mapa[x1][y1+i]!=0)
                                return false;
                }
                return true;
        }
        else
        {
                if(y1 == y2)
                {
                        if(x1==x2+1||x1==x2-1)
                                return true;
                        int alpha;
                        if(x1<x2)
                                alpha = 1;
                        else
                                alpha = -1;
                        for(int i=alpha; Math.abs(i)<Math.abs(x2-x1); i+= alpha)
                        {
```

```
                                        if(mapa[x1+i][y1]!=0)
                                                return false;
                                }
                                return true;
                        }
                        else
                        {
                                if(Math.abs((x2-x1)/(float)(y2-y1))!=1)
                                        return false;
                                int alphax;
                                int alphay;
                                if(x1<x2)
                                        alphax = 1;
                                else
                                        alphax = -1;
                                if(y1<y2)
                                        alphay = 1;
                                else
                                        alphay = -1;
                                int i, j;
                                for(i=x1+alphax, j=y1+alphay; Math.abs(i)<Math.abs(x2-
x1)&&Math.abs(j)<Math.abs(y2-y1)&&i>=0&&j>=0&&j<tam&&i<tam; i+= alphax, j+= alphay)
                                {
                                        if(mapa[i][j]!=0)
                                                return false;
                                }
                                if(i<0||j<0||i>tam||j>tam)
                                        return false;
                                return true;
                        }
                }
        }

        public boolean Validar_Movimiento(int x1, int y1, int x2, int y2, int acc)
        {

        if((x1>tam)||(x2>tam)||(y1>tam)||(y2>tam)||(x1<0)||(x2<0)||(y1<0)||(y2<0))
                        return false;
                if(mapa[x1][y1]==0)
                {
                        return false;
                }
                if(!Conectados(x1, y1, x2, y2))
                {
                        return false;
                }
                if(mapa[x1][y1]*mapa[x2][y2]>0)
                {
                        return false;
                }
                if(mapa[x1][y1]==1)
                {
                        if(Math.sqrt((posxReina2-x1)*(posxReina2-x1)+(posyReina2-
y1)*(posyReina2-y1))>=Math.sqrt((posxReina2-x2)*(posxReina2-x2)+(posyReina2-
y2)*(posyReina2-y2)))
                        {
                                return true;
                        }
                }
                if(mapa[x1][y1]==-1)
                {
                        if(Math.sqrt((posxReina1-x1)*(posxReina1-x1)+(posyReina1-
y1)*(posyReina1-y1))>=Math.sqrt((posxReina1-x2)*(posxReina1-x2)+(posyReina1-
y2)*(posyReina1-y2)))
                        {
                                return true;
```

```
                }
        }
        if((Math.abs(mapa[x1][y1])>2)||(acc==0))
                return true;
        return false;
}

public int Devuelve_Dimension()
{
        return tam;
}

public int Devuelve_Jugador()
{
        return jugador;
}

public void Cambia_Jugador()
{
        jugador = 1-jugador;
}

public int Devuelve_PosXReina1()
{
        return posxReina1;
}

public int Devuelve_PosYReina1()
{
        return posyReina1;
}

public int Devuelve_PosXReina2()
{
        return posxReina2;
}

public int Devuelve_PosYReina2()
{
        return posyReina2;
}

public int Devuelve_Valor(int x, int y)
{
        return mapa[x][y];
}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 * Esta funci—n muestra por pantalla el campo de juego,  *
 * representando con un # un mono del jugador 1 y @ un   *
 * mono del jugador 2, adem‡s de reprentar a las reinas  *
 * N* y N¼, respectivamente                              *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * */

public void Imprime_CampoJuego()
{
        System.out.print("\n   |");
        for(int i=0; i<tam; i++)
        {
                if(i<10)
                        System.out.format(" %d |", i);
                else
                        System.out.format(" %d|", i);
        }
        for(int i=0; i<tam; i++)
        {
```

```
                        if(i<10)
                                System.out.format("\n %d |", i);
                        else
                                System.out.format("\n %d|", i);
                        for(int j=0; j<tam; j++)
                        {
                                if(mapa[i][j]<0)
                                {
                                        if(mapa[i][j] == -1)
                                                System.out.print(" $ ");
                                        else
                                        {
                                                if(Math.abs(mapa[i][j])<10)
                                                        System.out.format("
%dº",Math.abs(mapa[i][j]));
                                                else

        System.out.format("%dº",Math.abs(mapa[i][j]));
                                        }
                                }
                                else
                                {
                                        if(mapa[i][j]>0)
                                        {
                                                if(mapa[i][j] == 1)
                                                        System.out.print(" # ");
                                                else
                                                {
                                                        if(Math.abs(mapa[i][j])<10)
                                                                System.out.format("
%d*",Math.abs(mapa[i][j]));
                                                        else

        System.out.format("%d*",Math.abs(mapa[i][j]));
                                                }
                                        }
                                        else
                                        {
                                                System.out.print(" 0 ");
                                        }
                                }
                                System.out.print("|");
                        }
                }
                System.out.print("\n");
                System.out.print("\n");
                System.out.print("\n");
        }

        /* * * * * * * * * * * * * * * * * * * * * * * * * * * *
         *  Esta funci—n lleva a cabo la jugada solicitada,  *
         *  modificando el mapa de juego y las variables     *
         *  internas, adem‡s de cambiar de jugador.                    *
         * * * * * * * * * * * * * * * * * * * * * * * * * * * */

        public void Hacer_Jugada(MQ_Jugada state)
        {
                MQ_Jugada jugada = state;
                int valor;
                int reina;
                if(jugador == 0)
                        valor = -1;
                else
                        valor = 1;
                if((jugada.Pos_x2_JUG1==posxReina1)&&(jugada.Pos_y2_JUG1==posyReina1))
                {
```

```
                        posxReina1=-1;
                        posyReina1=-1;
                }
                if((jugada.Pos_x2_JUG1==posxReina2)&&(jugada.Pos_y2_JUG1==posyReina2))
                {
                        posxReina2=-1;
                        posyReina2=-1;
                }
                if(Math.abs(mapa[jugada.Pos_x1_JUG1][jugada.Pos_y1_JUG1])>1)
                {
                        reina = mapa[jugada.Pos_x1_JUG1][jugada.Pos_y1_JUG1];
                        if(jugada.Acc_JUG1==1)
                        {
                                mapa[jugada.Pos_x1_JUG1][jugada.Pos_y1_JUG1] = -valor;
                                mapa[jugada.Pos_x2_JUG1][jugada.Pos_y2_JUG1] =
reina+valor;
                        }
                        else
                        {
                                mapa[jugada.Pos_x1_JUG1][jugada.Pos_y1_JUG1] = 0;
                                mapa[jugada.Pos_x2_JUG1][jugada.Pos_y2_JUG1] = reina;
                        }
                        if(jugador==0)
                        {
                                posxReina1=jugada.Pos_x2_JUG1;
                                posyReina1=jugada.Pos_y2_JUG1;
                        }
                        else
                        {
                                posxReina2=jugada.Pos_x2_JUG1;
                                posyReina2=jugada.Pos_y2_JUG1;
                        }
                }
                else
                {
                        if(Math.abs(mapa[jugada.Pos_x1_JUG1][jugada.Pos_y1_JUG1])==1)
                        {
                                mapa[jugada.Pos_x1_JUG1][jugada.Pos_y1_JUG1] = 0;
                                mapa[jugada.Pos_x2_JUG1][jugada.Pos_y2_JUG1] = -valor;
                        }
                }
                return;
        }
}
```

MQ_Heuristica.java

```
package monkeyqueen;

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 * Esta clase se encarga de manejar las cuatro distintas heur'sticas.  *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

public class MQ_Heuristica
{
	/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
	 *                                                                       *
	 * Funci—n que se encarga de manejar las heur'sticas                     *
	 *                                                                       *
	 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
	public float getHeuristicValue (MQ_CampoJuego campo, int heuristica, int
jugador)
	{
		if(heuristica==0)
			return (float)Math.random()%100-50;
		if(isGoalState(campo, jugador))
			return 10000000;
		if(isLooserState(campo, jugador))
			return -1000000;
		// Se envía a la función de heurística deseada heuristicaX()
		switch(heuristica)
		{
		case 1:
			return heuristica1(campo);
		case 2:
			return heuristica2(campo);
		case 3:
			return heuristica3(campo);
		case 4:
			return heuristica4(campo);
		default:
			return (float)Math.random()%100-50;
		}
	}
	/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
	 * Rodeamos a la reina enemiga:                                          *
	 *                                                                       *
	 * (NMonos aliados - NMonos enemigos + 1)/Dist Med a la reina enemiga    *
	 *                                                                       *
	 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

	public float heuristica1(Object state)
	{
		MQ_CampoJuego campo = (MQ_CampoJuego)state;
		int naliados = 0;
		int nenemi = 0;
		int dist = 0;
		int valor;
		int posxReina;
		int posyReina;
		if(campo.Devuelve_Jugador()==0)
		{
			posxReina = campo.Devuelve_PosXReina2();
			posyReina = campo.Devuelve_PosYReina2();
```

```
                        valor = 1;
                }
                else
                {
                        posxReina = campo.Devuelve_PosXReina1();
                        posyReina = campo.Devuelve_PosYReina1();
                        valor = -1;
                }
                for(int i=0; i<campo.Devuelve_Dimension(); i++)
                {
                        for(int j=0; j<campo.Devuelve_Dimension(); j++)
                        {
                                int actual = campo.Devuelve_Valor(i, j)*valor;
                                if(actual<0)
                                {
                                        nenemi++;
                                }
                                if(actual>0)
                                {
                                        naliados++;
                                        dist += Math.sqrt((i-posxReina)*(i-posxReina) +
(j-posyReina)*(j-posyReina));
                                }
                        }
                }
                dist /= (float)naliados;
                if(naliados>3)
                        return (naliados - nenemi + 1)/(float)dist;
                else
                {
                        int posatac = 0;
                        for(int i=0; i<campo.Devuelve_Dimension(); i++)
                        {
                                for(int j=0; j<campo.Devuelve_Dimension(); j++)
                                {
                                        int actual = campo.Devuelve_Valor(i, j)*valor;
                                        if(actual<0)
                                        {
                                                if(campo.Conectados(i, j, posxReina,
posyReina))
                                                        posatac++;
                                        }
                                }
                        }
                        if(posatac==0)
                                return (naliados - nenemi + 1);
                        return (naliados - nenemi + 1)/(float)posatac;
                }
        }

        /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
         * Evitamos cercos a la reina aliada:
                          *
         *
                                                                *
         * (NMonos aliados - NMonos enemigos + 1)*Dist Med a la reina aliada     *
         *
                                                                *
         * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

        public float heuristica2(Object state)
        {
                MQ_CampoJuego campo = (MQ_CampoJuego)state;
                int naliados = 0;
                int nenemi = 0;
                int dist = 0;
```

```
		int valor;
		int posxReina;
		int posyReina;
		if(campo.Devuelve_Jugador()==0)
		{
			posxReina = campo.Devuelve_PosXReina1();
			posyReina = campo.Devuelve_PosYReina1();
			valor = 1;
		}
		else
		{
			posxReina = campo.Devuelve_PosXReina2();
			posyReina = campo.Devuelve_PosYReina2();
			valor = -1;
		}
		for(int i=0; i<campo.Devuelve_Dimension(); i++)
		{
			for(int j=0; j<campo.Devuelve_Dimension(); j++)
			{
				int actual = campo.Devuelve_Valor(i, j)*valor;
				if(actual<0)
				{
					nenemi++;
					dist += Math.sqrt((i-posxReina)*(i-posxReina) +
(j-posyReina)*(j-posyReina));
				}
				if(actual>0)
				{
					naliados++;
				}
			}
		}
		dist /= (float)nenemi;
		return (naliados - nenemi + 1)*dist;
	}

	/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
	 * Posiblidad de ataque enemigo:
	                    *
	 *
	                                                    *
	 * (NMonos aliados - NMonos enemigos + 1)/(NPos atac reina aliada + 1)   *
	 *
	                                                    *
	 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
	public float heuristica3(Object state)
	{
		MQ_CampoJuego campo = (MQ_CampoJuego)state;
		int naliados = 0;
		int nenemi = 0;
		int posatac = 1;
		int valor;
		int posxReina;
		int posyReina;
		if(campo.Devuelve_Jugador()==0)
		{
			posxReina = campo.Devuelve_PosXReina1();
			posyReina = campo.Devuelve_PosYReina1();
			valor = 1;
		}
		else
		{
			posxReina = campo.Devuelve_PosXReina2();
			posyReina = campo.Devuelve_PosYReina2();
			valor = -1;
		}
```

```
		for(int i=0; i<campo.Devuelve_Dimension(); i++)
		{
			for(int j=0; j<campo.Devuelve_Dimension(); j++)
			{
				int actual = campo.Devuelve_Valor(i, j)*valor;
				if(actual>0)
				{
					naliados++;
				}
				else
				{
					if(actual<0)
					{
						nenemi++;
						if(campo.Conectados(i, j, posxReina,
posyReina))
							posatac++;
					}
				}
			}
		}
		return (naliados - nenemi + 1)/(float)posatac;
	}

	/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
	 * Posiblidad de ataque aliado:
	                              *
	 *
	                                                    *
	 * (Aliados - Enemigos + 1)*NPos atac reina enemiga
	                 *
	 *
	                                                    *
	 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
	public float heuristica4(Object state)
	{
		MQ_CampoJuego campo = (MQ_CampoJuego)state;
		int naliados = 0;
		int nenemi = 0;
		int posatac = 0;
		int valor;
		int posxReina;
		int posyReina;
		if(campo.Devuelve_Jugador()==0)
		{
			posxReina = campo.Devuelve_PosXReina2();
			posyReina = campo.Devuelve_PosYReina2();
			valor = 1;
		}
		else
		{
			posxReina = campo.Devuelve_PosXReina1();
			posyReina = campo.Devuelve_PosYReina1();
			valor = -1;
		}
		for(int i=0; i<campo.Devuelve_Dimension(); i++)
		{
			for(int j=0; j<campo.Devuelve_Dimension(); j++)
			{
				int actual = campo.Devuelve_Valor(i, j)*valor;
				if(actual<0)
				{
					nenemi++;
				}
				else
				{
```

```
                                if(actual>0)
                                {
                                        naliados++;
                                        if(campo.Conectados(i, j, posxReina,
posyReina))
                                                posatac++;
                                }
                        }
                }
        }
        return (naliados - nenemi + 1)*posatac;
    }


    /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
     * isGoalState:
                                                *
     *
                                                        *
     * Comprueba si estamos ante una jugada ganadora
           *
     *
                                                        *
     * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

    public boolean isGoalState(Object state, int jugador)
    {
        MQ_CampoJuego campo = (MQ_CampoJuego) state;
        if(jugador==0)
        {
                if(campo.Devuelve_PosXReina2()==-
1&&campo.Devuelve_PosYReina2()==-1)
                        return true;
        }
        else
        {
                if(campo.Devuelve_PosXReina1()==-
1&&campo.Devuelve_PosYReina1()==-1)
                        return true;
        }
        return false;
    }

    /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
     * isLooserState:
                                           *
     *
                                                        *
     * Comprueba si estamos ante una jugada perdedora
           *
     *
                                                        *
     * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

    public boolean isLooserState(Object state, int jugador)
    {
        MQ_CampoJuego campo = (MQ_CampoJuego) state;
        int posxReina, posyReina, valor;
        if(jugador==0)
        {
                if(campo.Devuelve_PosXReina1()==-
1&&campo.Devuelve_PosYReina1()==-1)
                        return true;

                posxReina = campo.Devuelve_PosXReina1();
                posyReina = campo.Devuelve_PosYReina1();
```

```
                        valor = 1;
                }
                else
                {
                        if(campo.Devuelve_PosXReina2()==-
1&&campo.Devuelve_PosYReina2()==-1)
                                return true;

                        posxReina = campo.Devuelve_PosXReina2();
                        posyReina = campo.Devuelve_PosYReina2();
                        valor = -1;
                }
                if(campo.Conectados(campo.Devuelve_PosXReina1(),
campo.Devuelve_PosYReina1(), campo.Devuelve_PosXReina2(),
campo.Devuelve_PosYReina2()))
                        return true;
                for(int i=0; i<campo.Devuelve_Dimension(); i++)
                {
                        for(int j=0; j<campo.Devuelve_Dimension(); j++)
                        {
                                if(campo.Devuelve_Valor(i, j)*valor<0)
                                {
                                        if (campo.Conectados(i,j,posxReina,posyReina))
                                        {
                                                return true;
                                        }
                                }
                        }
                }
                return false;
        }
}
```

MQ_Juego.java

```
package monkeyqueen;

import java.util.ArrayList;
import java.util.List;

	/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
	 * Esta clase se encarga de manejar el flujo de juego,   *
	 * obteniendo las distintas jugadas y calculando el      *
	 * estado actual de cada fase de juego.                  *
	 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

public class MQ_Juego
{
	private MQ_CampoJuego campo;
	private int prof1;
	private int prof2;
	private int heuristica1;
	private int heuristica2;
	private int tiporesol1;
	private int tiporesol2;

	/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
	 * Esta es la funci—n que inicializa el juego.                    *
	 * Parametros:
	 *
	 *     mapa[][]: mapa inicial del juego                                  *
	 *  dim: dimensi—n del cuadrado de juego                              *
	 *  nuevo: si el mapa ya esta definidio    (1 == si)                  *
	 *  tamX: tama–o de la pila de la reina X                             *
	 *  profuX: profundidad a la que puede bajar el jugador X  *
	 *  heurX: heuristica que utilizara el jugador X            *
	 *  aiX: Si el jugador es humano (0) o m‡quina (1)          *
	 *  resolX: tipo de resolucion del arbol expandido          *
	 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

	public MQ_Juego(int[][] mapa, int dim, int nuevo, int tam1, int profu1, int
heur1, int resol1,

			int tam2, int profu2, int heur2, int resol2)
	{
		if(nuevo==1)
			campo = new MQ_CampoJuego(mapa, dim);
		else
			campo = new MQ_CampoJuego(dim, dim/2, 0, tam1, (dim/2)+1, dim-
1, tam2);
		prof1 = profu1;
		prof2 = profu2;
		heuristica1 = heur1;
		heuristica2 = heur2;
		tiporesol1 = resol1;
		tiporesol2 = resol2;
	}

	public MQ_Juego(int dim, int ia1, int ia2, int tam1, int tam2, int profu1, int
profu2, int resol1, int resol2,
				int posx1, int posy1, int posx2, int posy2, int
jugador)
	{
		campo = new MQ_CampoJuego(dim, posx1, posy1, tam1, posx2, posy2,
tam2);
		prof1 = profu1;
		prof2 = profu2;
		heuristica1 = ia1;
```

```
		heuristica2 = ia2;
		tiporesol1 = resol1;
		tiporesol2 = resol2;
		if(jugador==1)
			campo.Cambia_Jugador();
	}

	public void Imprime()
	{
		campo.Imprime_CampoJuego();
	}

	public int ObtenerFichaJugador(int x, int y)
	{
		int tam = campo.Devuelve_Dimension();
		if((x>tam)||(x<0)||(y>tam)||(y<0))
			return -10;
		int m = campo.Devuelve_Valor(x, y);
		if(m<0)
			return 1;
		else
		{
			if(m>0)
				return 0;
			else
				return -1;
		}
	}

	public boolean Fin_ganador()
	{
		if((campo.Devuelve_PosXReina2()==-1)&&(campo.Devuelve_PosYReina2()==-
1))
			return true;
		return false;
	}

	public boolean Fin_perdedor()
	{
		if((campo.Devuelve_PosXReina1()==-1)&&(campo.Devuelve_PosYReina1()==-
1))
			return true;
		return false;
	}

	public int CompruebaFin()
	{
		if(Fin_ganador())
			return 1;
		if(Fin_perdedor())
			return -1;
		return 0;
	}

	/* * * * * * * * * * * * * * * * * * * * * * * * * * * * *
	 * Esta funcion lleva a cabo todas las comprobaciones    *
	 * y llamadas pertinentes para cada jugada automatica.    *
	 * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

	public void Juega(int jugador, MQ_Partida partida)
	{
		List<MQ_Jugada> posibles = new ArrayList<MQ_Jugada>();;
		int valor;
		if(jugador == 0)
			valor = 1;
		else
```

```
                        valor = -1;
                int tam = campo.Devuelve_Dimension();
                for(int i=0; i<tam; i++)
                {
                        for(int j=0; j<tam; j++)
                        {
                                if(campo.Devuelve_Valor(i, j)*valor>0)
                                {
                                        posibles.addAll(Posibles_Jugadas(i, j, campo));
                                }
                        }
                }
                if(posibles.size()==0)
                {
                        campo.tablas = true;
                        return;
                }
                MQ_Jugada jugada = Mejor_Jugada(posibles, jugador);
                campo.Hacer_Jugada(jugada);
                partida.Inserta_Jugada(jugada);
                campo.Cambia_Jugador();
                return;
        }

        public boolean Comprueba_Jugada(MQ_Jugada jugada)
        {
                return campo.Validar_Movimiento(jugada.Pos_x1_JUG1,
jugada.Pos_y1_JUG1, jugada.Pos_x2_JUG1, jugada.Pos_y2_JUG1, jugada.Acc_JUG1);
        }

        public void Juega_Jugada(MQ_Jugada jugada)
        {
                campo.Hacer_Jugada(jugada);
                campo.Cambia_Jugador();
        }

        public void Cambia_Jugador()
        {
                campo.Cambia_Jugador();
        }

        /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
         * Esta funcion devuelve todas las posibles jugadas de la  *
         * ficha actual con el jugador actual, comprobando que las *
         * restricciones del juego se cumplan.                      *
         * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

        public List<MQ_Jugada> Posibles_Jugadas(int x1, int y1, MQ_CampoJuego
campodejuego)
        {
                List<MQ_Jugada> posibles = new ArrayList<MQ_Jugada>();
                int tam = campodejuego.Devuelve_Dimension();
                for(int i=0; i<tam; i++)
                {
                        for(int j=0; j<tam; j++)
                        {
                                if((x1==i)&&(y1==j))
                                {
                                        continue;
                                }
                                if(campodejuego.Conectados(x1, y1, i, j))
                                {
                                        int xa = campodejuego.Devuelve_Valor(x1, y1);
                                        int xb = campodejuego.Devuelve_Valor(i, j);
                                        if((xa*xb>0)) // -x*-x == x*x != -x*x
                                        {
```

```
                                        continue;
                                }
                                else
                                {
                                        MQ_Jugada jugada;
                                        if(xb!=0)
                                        {
                                                if(!(campo.Validar_Movimiento(x1,
y1, i, j, 0)))
                                                {
                                                        continue;
                                                }
                                                jugada = new MQ_Jugada(x1, i, y1,
j, 0);
                                        }
                                        else
                                        {
                                                if(!(campo.Validar_Movimiento(x1,
y1, i, j, 1)))
                                                {
                                                        continue;
                                                }
                                                jugada = new MQ_Jugada(x1, i, y1,
j, 1);
                                        }
                                        posibles.add(jugada);
                                }
                        }
                }
        }
        return posibles;
    }

    /* * * * * * * * * * * * * * * * * * * * * * * * * * * * *
     * Esta funcion devuelve la jugada que minimice la       *
     * heur’stica, es decir, que tienda a vencer al enemigo  *
     * en funci—n a la heuristica escogida.                                *
     * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

    public boolean Empatado()
    {
        return campo.tablas;
    }

    public MQ_Jugada Mejor_Jugada(List<MQ_Jugada> todas, int jugador)
    {
        if(todas.size()==0)
        {
                return null;
        }
        int profundidad = (jugador == 0) ? prof1 : prof2;
        if(profundidad==0)
        {
                return todas.get((int) (Math.random()%todas.size()));
        }
        for(int i=0; i<todas.size(); i++)
        {
                MQ_CampoJuego aux = new MQ_CampoJuego(campo);
                aux.Hacer_Jugada(todas.get(i));
                MQ_Heuristica heuris = new MQ_Heuristica();
                float k;
                if(jugador==0)
                        k = heuris.getHeuristicValue(aux, heuristica1, jugador);
                else
                        k = heuris.getHeuristicValue(aux, heuristica2, jugador);
                if(k==10000000)
```

```
                        return todas.get(i);
                if(k==-1000000)
                {
                        /*
                         * ATENCION !!! EN CASO DE NO HABER MOVIMIENTO POSIBLE
SE USA EL òLTIMO
                         */
                        if(todas.size()>1)
                        {
                                todas.remove(i);
                                i--;
                        }
                        continue;
                }
        }
        int resol = (jugador == 0) ? tiporesol1 : tiporesol2;
        int herencias[][][] = new int[profundidad][570][19000];
    for(int i=0; i<todas.size(); i++)
                herencias[0][0][i] = 1;
        int i=1;
        int valor = (jugador == 0) ? 1 : -1;
        List<MQ_Jugada> posibles = todas;
        MQ_CampoJuego camposheredados[][] = new
MQ_CampoJuego[profundidad][570];
        int hijazos[] = new int[profundidad];
        MQ_CampoJuego campojuego = new MQ_CampoJuego(campo);
        int total = todas.size();
        hijazos[0] = total;
        // Bucle que lleva a cabo la obtenci—n del vector base del ‡rbol
        while(i<profundidad)
        {
                List<MQ_Jugada> auxiliar = new ArrayList<MQ_Jugada>();
                int numaux = 0;
                for(int j=0; j<posibles.size(); j++)
                {
                        MQ_CampoJuego aux = new MQ_CampoJuego(campojuego);
                        aux.Hacer_Jugada(posibles.get(j));
                        MQ_CampoJuego campoaux = new MQ_CampoJuego(aux);
                        for(int k=0; k<campoaux.Devuelve_Dimension(); k++)
                        {
                                for(int l=0; l<campoaux.Devuelve_Dimension();
l++)
                                {
                                        herencias[i][j][auxiliar.size()] = 0;
                                        if(campoaux.Devuelve_Valor(k, l)*valor>0)
                                        {
                                                int naux = auxiliar.size();
                                                auxiliar.addAll(Posibles_Jugadas(k,
l, campoaux));
                                                if(auxiliar.size()!=0)
                                                {
                                                        for(; naux<auxiliar.size();
naux++)
                                                        {

        herencias[i][j][naux-numaux] = 1;
                                                        }
                                                }
                                        }
                                }
                        }
                        numaux = auxiliar.size();
                        camposheredados[i][j] = campoaux;
                }
                hijazos[i] = auxiliar.size();
                posibles = auxiliar;
```

```
                        valor *= -1;
                        i++;
                }
                // Creamos el vector con las heur’sticas de la base del ‡rbol
                float valoresh[] = new float[posibles.size()+1];
                int padresherencia = (profundidad == 1) ? 0 : hijazos[profundidad-2];
                int j=0;
                for(i=0; i<padresherencia; i++)
                {
                        int k=0;
                        while(herencias[profundidad-1][i][k]!=0)
                        {
                                MQ_Heuristica heuris = new MQ_Heuristica();
                                MQ_CampoJuego aux = new
MQ_CampoJuego(camposheredados[profundidad-1][i]);
                                aux.Hacer_Jugada(posibles.get(j));
                                if(campo.Devuelve_Jugador()==0)
                                        valoresh[j] = heuris.getHeuristicValue(aux,
heuristica1, jugador);
                                else
                                        valoresh[j] = heuris.getHeuristicValue(aux,
heuristica2, jugador);
                                j++;
                                k++;
                        }
                }
                // Subimos por el ‡rbol, llevando a cabo la estrategia definida
(minimax o alfa-beta)

                /*
                 * MINIMAX
                 */
                int finalista = 0;
                if(resol == 0)
                {
                        total = posibles.size();
                        for(i=profundidad-2; i>=0; i--)
                        {
                                for(j=0; j<hijazos[i]; j++)
                                {
                                        int k=0;
                                        float bueno = (i%2==0) ? -1000000 : 1000000;
                                        int m=0;
                                        while(herencias[i][j][k]==1)
                                        {
                                                if(i%2==1) // Es impar -> minimizamos
                                                {
                                                        if(valoresh[m]<bueno)
                                                        {
                                                                bueno = valoresh[m];
                                                                finalista = m;
                                                        }
                                                        else
                                                        {
                                                                if(valoresh[m]==bueno)
                                                                {

        if(Math.random()%2==0)

                                                                        {
                                                                                bueno =
valoresh[m];

                                                                                finalista =
m;

                                                                        }
                                                                }
                                                        }
```

```
                                        }
                                        else // es par -> maximizamos
                                        {
                                                if(valoresh[m]>bueno)
                                                {
                                                        bueno = valoresh[m];
                                                        finalista = m;
                                                }
                                                else
                                                {
                                                        if(valoresh[m]==bueno)
                                                        {

        if(Math.random()%2==0)
                                                                {
                                                                        bueno =
valoresh[m];
                                                                        finalista =
m;
                                                                }
                                                        }
                                                }
                                        }
                                        m++;
                                        k++;
                                }
                                valoresh[j] = bueno;
                        }
                }
        }

        /*
         * ALFABETA
         */
        if(resol == 1)
        {
                int[] inicio = new int[profundidad];
                for(i=0; i<profundidad; i++)
                        inicio[i] = 0;
                finalista = (int)alpha_beta(herencias, hijazos, valoresh, 0, 0,
profundidad, -10000000, 1000000, true, inicio);
        }
        return todas.get(finalista);
    }

    public float alpha_beta(int[][][] herencias, int hijazos[], float[]
heuristicas, int prof_actual, int hijo_actual,
                                                int profundidad, float alpha, float
beta, boolean padre, int[] vinicio)
    {
        float var;
        if(profundidad==prof_actual||padre==false) // Nodo hoja
        {
                var = heuristicas[0];
                return var;
        }
        if(profundidad==prof_actual+1) // Nodo padre final
        {
                int finalista = 0;
                for(int i=0; i<hijazos[0]; i++)
                {
                        float[] h = new float[1];
                        h[0] = heuristicas[i];
                        var = alpha_beta(herencias, hijazos, h, prof_actual+1,
i, profundidad, alpha, beta, false, vinicio);
                        if(prof_actual%2==0)
```

```
                        {
                                if(alpha<var)
                                {
                                        alpha = var;
                                        finalista = i;
                                }
                                else
                                        if(alpha==var)
                                                if(Math.random()%2==1)
                                                {
                                                        alpha = var;
                                                        finalista = i;
                                                }
                                if(alpha>=beta)
                                {
                                        if(prof_actual == 0)
                                        {
                                                return finalista;
                                        }
                                        return alpha;
                                }
                        }
                        else
                        {
                                if(beta>var)
                                {
                                        beta = var;
                                        finalista = i;
                                }
                                else
                                        if(beta==var)
                                                if(Math.random()%2==1)
                                                {
                                                        beta = var;
                                                        finalista = i;
                                                }
                                if(alpha>=beta)
                                {
                                        if(prof_actual == 0)
                                        {
                                                return finalista;
                                        }
                                        return beta;
                                }
                        }
                }
                if(prof_actual == 0)
                {
                        return finalista;
                }
                if(prof_actual%2==0)
                {
                        return alpha;
                }
                else
                {
                        return beta;
                }
        }

        int[] inicio0 = new int[profundidad-prof_actual];
        inicio0 = vinicio;
        int finalista = 0;
        for(int m = 0; m<hijazos[0]; m++)
        {
                int[] inicio = new int[profundidad-prof_actual];
```

```
                        for(int i=0; i<profundidad-prof_actual; i++)
                                inicio[i] = inicio0[i];
                        int[] hijos = new int[profundidad-prof_actual];
                        int numpadritos = 1;
                        for(int i=prof_actual+1; i<profundidad; i++)
                        {
                                int padrito = vinicio[i];
                                int num = 0;
                                for(int n=padrito; n<numpadritos; n++)
                                {
                                        int k=0;

        while(herencias[i][inicio0[i]+n][k]==1&&k<hijazos[i-prof_actual])
                                        {
                                                num++;
                                                k++;
                                        }
                                }
                                numpadritos = num;
                                inicio[i-prof_actual] += numpadritos+1;
                                hijos[i-prof_actual-1] = numpadritos+1;
                                padrito = inicio0[i-prof_actual];
                        }
                        float[] heuristics = new float[inicio[profundidad-prof_actual-
1]-inicio0[profundidad-prof_actual-1]];
                        for(int i=inicio0[profundidad-prof_actual-1];
i<inicio[profundidad-prof_actual-1]; i++)
                        {
                                heuristics[i-inicio0[profundidad-prof_actual-1]] =
heuristicas[i];
                        }
                        if(profundidad-prof_actual==1)
                                var = alpha_beta(herencias, hijos, heuristics,
prof_actual+1, m, profundidad, alpha, beta, false, inicio);
                        else
                                var = alpha_beta(herencias, hijos, heuristics,
prof_actual+1, m, profundidad, alpha, beta, true, inicio);
                        if(prof_actual%2==0)
                        {
                                if(alpha<var)
                                {
                                        alpha = var;
                                        finalista = m;
                                }
                                else
                                        if(alpha==var)
                                                if(Math.random()%2==1)
                                                {
                                                        alpha = var;
                                                        finalista = m;
                                                }
                                if(alpha>=beta)
                                {
                                        if(prof_actual == 0)
                                        {
                                                return finalista;
                                        }
                                        return alpha;
                                }
                        }
                        else
                        {
                                if(beta>var)
                                {
                                        beta = var;
                                        finalista = m;
```

```
                        }
                        else
                                if(beta==var)
                                        if(Math.random()%2==1)
                                        {
                                                beta = var;
                                                finalista = m;
                                        }
                                if(alpha>=beta)
                                {
                                        if(prof_actual == 0)
                                        {
                                                return finalista;
                                        }
                                        return beta;
                                }
                        }
                        inicio0 = inicio;
                }
                if(prof_actual == 0)
                {
                        return finalista;
                }
                if(prof_actual%2==0)
                {
                        return alpha;
                }
                else
                {
                        return beta;
                }
        }
}
```

MQ_Jugada.java

```
package monkeyqueen;

	/* * * * * * * * * * * * * * * * * * * * * * * * * * * *
	 * Esta clase guarda una jugada de la partida,       *
	 * siendo una jugada el movimiento del jugador 1.    *
	 * Si en Acci—n tenemos un 1, es un movimiento            *
	 * simple, si tenemos un 0, es un ataque.                 *
	 * * * * * * * * * * * * * * * * * * * * * * * * * * * */

public class MQ_Jugada
{
	/* * * * * * * * * * * * * * * * * * * * * * * * * * * *
	 * Las variables de la clase muestran las posiciones *
	 * iniciales y finales de las fichas a mover y la    *
	 * acci—n realizada por cada jugador.                *
	 * * * * * * * * * * * * * * * * * * * * * * * * * * * */

	public int Pos_x1_JUG1;
	public int Pos_x2_JUG1;
	public int Pos_y1_JUG1;
	public int Pos_y2_JUG1;
	public int Acc_JUG1;

	public MQ_Jugada(int px1, int px2, int py1, int py2, int ph)
	{
		Pos_x1_JUG1 = px1;
		Pos_x2_JUG1 = px2;
		Pos_y1_JUG1 = py1;
		Pos_y2_JUG1 = py2;
		Acc_JUG1 = ph;
	}

}
```

MQ_Partida.java

```
package monkeyqueen;

import java.io.*;
import java.util.ArrayList;
import java.util.List;

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 * Esta clase representa el registro de la partida a lo  *
 * largo del tiempo. Permite guardar su estado en un     *
 * fichero o reutilizar una partida anterior a partir    *
 * de un fichero ya existente.                           *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

public class MQ_Partida
{
	private List<MQ_Jugada> Partida;
	private int nJugadas;
	private int[] valores_iniciales;
	private String ruta;

	/* * * * * * * * * * * * * * * * * * * * * * * * * * *
	 * Constructor de la clase. Inicialmente est‡ vacio. *
	 * * * * * * * * * * * * * * * * * * * * * * * * * * */

	public MQ_Partida()
	{
		Partida = new ArrayList<MQ_Jugada>();
		nJugadas = 0;
		valores_iniciales = new int[13];
	}

	/* * * * * * * * * * * * * * * * * * * * * * *
	 * Inserta una jugada en la partida completa *
	 * * * * * * * * * * * * * * * * * * * * * * */

	public void Set_Ruta(String Ruta)
	{
		ruta = Ruta;
	}

	public String Get_ruta()
	{
		return ruta;
	}

	public void Inserta_Jugada(MQ_Jugada Jugada)
	{
		Partida.add(Jugada);
		nJugadas++;
		return;
	}

	/* * * * * * * * * * * * * * * * * * * * * * * * * * * * *
	 * Guarda la partida en el fichero cuya ruta     *
	 * se le pasa por par‡metro. Devuelve el tama–o  *
	 * del fichero. Si no hay jugadas, se devuelve     *
	 * un -1. En caso de fallo, -1.                          *
	 * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

	public int Salva_Partida()
	{
		FileWriter fichero = null;
		PrintWriter pw = null;
```

```
              try
              {
                     fichero = new FileWriter(ruta);
                     pw = new PrintWriter(fichero);

                     pw.println(valores_iniciales[0] + " " + valores_iniciales[1] +
" " + valores_iniciales[2] + " " + valores_iniciales[3]
                                                                                  + " "
+ valores_iniciales[4] + " " + valores_iniciales[5] + " " + valores_iniciales[6]
                                                                                  + " "
+ valores_iniciales[7] + " " + valores_iniciales[8] + " " + valores_iniciales[9]
                                                                                  + " "
+ valores_iniciales[10] + " " + valores_iniciales[11] + " " + valores_iniciales[12]);
                     pw.println(nJugadas);
                     for(int i=0; i<nJugadas; i++)
                     {
                            pw.println(Partida.get(i).Pos_x1_JUG1 + " " +
Partida.get(i).Pos_y1_JUG1 + " "
                                   + Partida.get(i).Pos_x2_JUG1 + " " +
Partida.get(i).Pos_y2_JUG1 + " " + Partida.get(i).Acc_JUG1);
                     }

              } catch (Exception e) {
                     return -1;
              } finally {
                     try {
                            // Nuevamente aprovechamos el finally para
                            // asegurarnos que se cierra el fichero.
                            if (null != fichero)
                            fichero.close();
              } catch (Exception e2) {
                  return -1;
              }
           }
              return 0 ;
       }

       /* * * * * * * * * * * * * * * * * * * * * * * * *
        * Carga la partida desde un fichero cuya ruta   *
        * se le pasa por par‡metro. Devuelve el nœmero  *
        * de jugadas hechas. En caso de no haber          *
        * partidas, se devolver‡ -2. En caso de           *
        * fallo, -1.                                                    *
        * * * * * * * * * * * * * * * * * * * * * * * * */

       public int[] ValoresIniciales()
       {
              return valores_iniciales;
       }

       public void SetValoresIniciales(int valores[])
       {
              valores_iniciales = valores;
       }

       public int NumeroJugadas()
       {
              return nJugadas;
       }

       public MQ_Jugada ObtenerJugada(int indice)
       {
              return Partida.get(indice);
       }

       public int Carga_Partida(String ruta)
```

```
        {
               File archivo = null;
            FileReader fr = null;
            BufferedReader br = null;

            try
            {
               // Apertura del fichero y creacion de BufferedReader para poder
               // hacer una lectura comoda (disponer del metodo readLine()).
               archivo = new File (ruta);
               fr = new FileReader (archivo);
               br = new BufferedReader(fr);

               // Lectura del fichero
               String linea;
            linea = br.readLine();
            String[] valores_init = linea.split(" ");
            valores_iniciales[0] = Integer.parseInt(valores_init[0]);
            valores_iniciales[1] = Integer.parseInt(valores_init[1]);
            valores_iniciales[2] = Integer.parseInt(valores_init[2]);
            valores_iniciales[3] = Integer.parseInt(valores_init[3]);
            valores_iniciales[4] = Integer.parseInt(valores_init[4]);
            valores_iniciales[5] = Integer.parseInt(valores_init[5]);
            valores_iniciales[6] = Integer.parseInt(valores_init[6]);
            valores_iniciales[7] = Integer.parseInt(valores_init[7]);
            valores_iniciales[8] = Integer.parseInt(valores_init[8]);
            valores_iniciales[9] = Integer.parseInt(valores_init[9]);
            valores_iniciales[10] = Integer.parseInt(valores_init[10]);
            valores_iniciales[11] = Integer.parseInt(valores_init[11]);
            valores_iniciales[12] = Integer.parseInt(valores_init[12]);
                nJugadas = Integer.parseInt(br.readLine());
                if(nJugadas==0)
                   return -2;
                for(int i=0; i<nJugadas; i++)
                {
                   linea = br.readLine();
                   String[] valores = linea.split(" ");
                   MQ_Jugada jugada = new MQ_Jugada(Integer.parseInt(valores[0]),

        Integer.parseInt(valores[1]),

        Integer.parseInt(valores[2]),

        Integer.parseInt(valores[3]),

        Integer.parseInt(valores[4]));
                   Partida.add(jugada);
                }
            }
            catch(Exception e){
               return -1;
            }finally{
               // En el finally cerramos el fichero, para asegurarnos
               // que se cierra tanto si todo va bien como si salta
               // una excepcion.
               try{
                  if( null != fr ){
                     fr.close();
                  }
               }catch (Exception e2){
                  return -1;
               }
            }
               return 0;
        }
}
```