

NOMBRE:
APELLIDOS:
NIA:
GRUPO:

2ª Parte: Problemas

Duración: 3 horas

Puntuación: 5 puntos (sobre 10)

Fecha: 12 de septiembre de 2005

No se podrán usar libros ni apuntes

Se debe obtener un mínimo de 2 puntos en esta parte para aprobar.

Para facilitar la comprensión del problema se incluye el código (incompleto) al final del enunciado.

Un tipo de juegos muy conocido son los relacionados con laberintos. En este problema vamos a implementar uno muy sencillo, cuyo objetivo es salir del laberinto (llegar a la casilla de salida) antes de que nos alcance uno de los perseguidores.

Descripción del juego:

El jugador mueve una casilla por turno, en horizontal o vertical (no está permitido mover en diagonal). Los perseguidores mueven 2 casillas por turno, automáticamente después de mover el jugador.

Casillas-trampa: En el laberinto hay casillas-trampa. Si el jugador cae en una trampa muere y termina la partida. Si cae un perseguidor, espera 3 turnos sin mover.

Muros: Algunas casillas tienen muros. Pueden estar arriba, abajo, a la izquierda o a la derecha de la casilla y puede haber varios muros en una misma casilla. Hay muros alrededor de todo el laberinto (en las casillas de los extremos, en la orientación correspondiente), para simplificar las comprobaciones de los movimientos. Los personajes no pueden moverse a través de los muros.



Ilustración 1 Pantalla del juego

Apartado 1:

Crea la interfaz gráfica del juego, de acuerdo a la imagen anterior. Para ello:

Completa la declaración de la clase **Laberinto**.

Completa la clase **Casilla**, que representa cada una de las casillas del laberinto y contiene información sobre su posición, si es una trampa o salida, si tiene muros, y métodos para poner y quitar personajes.

Programa el método **void crearLaberinto()** de la clase **Laberinto**:

Este método crea los objetos necesarios para el juego (casillas del laberinto, jugador, perseguidores), convenientemente inicializados. Utiliza un objeto de la clase **GestorConfiguracionXML**, que implementa el interfaz **InterfazConfiguracionXML**, para obtener los datos necesarios (número de filas y columnas del laberinto, qué casillas son trampa, cuál es la casilla de salida, dónde hay muros, dónde están situados inicialmente los personajes...).

Programa el método **void crearControles()** de la clase **Laberinto**:

Este método crea los botones que permiten controlar al jugador y prepara el escuchador que los atiende, de la clase **EscuchadorLaberinto**. Es decir, crea los botones y hace todo lo necesario para que el programa responda a su pulsación. Como se ve en la imagen, hay un botón para mover hacia **ARRIBA**, con el texto "**^**"; otro para mover hacia **ABAJO**, con el texto "**v**"; otro para mover hacia la **IZDA**, con el texto "**<**"; y otro para mover hacia la **DCHA**, con el texto "**>**". *Observa que los botones de control tienen texto, no imágenes. Utiliza UN ÚNICO objeto escuchador para los cuatro botones.*

Programa el método **void colocarElementosGraficos()** de la clase **Laberinto**:

Este método coloca los componentes gráficos del juego de acuerdo a la disposición de la figura y lo visualiza.

Apartado 2:

Programa la clase escuchadora de los botones que controlan el movimiento del jugador. Ten en cuenta que se utiliza UN ÚNICO objeto escuchador para los cuatro botones. Ten en cuenta también que cada vez que mueve el jugador (si se ha tenido éxito con el movimiento, es decir, ha podido mover en esa dirección), se mueven automáticamente después todos los perseguidores. Y que tendrás que comprobar si el juego finaliza.

Apartado 3:

Implementa el método **void mover()** de la clase **Perseguidor**, teniendo en cuenta que el movimiento de un perseguidor sigue las siguientes normas:

- 1º intenta acercarse al protagonista en horizontal
- Si no puede (porque hay un muro) o ya está en la misma columna que el protagonista, intenta moverse en vertical.
- Si no puede moverse tampoco en vertical (porque haya un muro), pasa su turno.
- El método debe comprobar:
 - si se ha alcanzado al protagonista, en cuyo caso se llama al método **finalizarJuego**
 - si se cae en una trampa, en cuyo caso debe esperar 3 turnos (invocar al método **caeEnTrampa()**)

Recuerda que:

- En cada turno, el perseguidor hace 2 movimientos (siguiendo para cada uno de ellos las normas anteriores).
- La posición de un **Personaje** (tanto **Perseguidor** como **Jugador**) se almacena como atributos del objeto (**posX** y **posY**).
- Hay muros alrededor de todo el laberinto, para simplificar las comprobaciones de los movimientos.

Apartado 4:

Programa el método `void finalizarJuego(boolean exito)` de la clase `Laberinto`.

Este método presenta un archivo de vídeo ("<http://www.it.uc3m.es/laberinto.mpg>") en la parte central de la ventana (donde antes aparecía el laberinto), que debe reproducirse de forma indefinida.

Además, presenta un cuadro de diálogo indicando que el juego ha terminado, con el mensaje adecuado según el jugador haya ganado (`exito true`) o perdido (`exito false`).

Por último, finaliza la aplicación cuando el usuario cierra el cuadro de diálogo.

No es necesario tratar las excepciones del paquete JMF, pero sí otras como las de entrada/salida...

Apartado 5:

Los datos del juego se almacenan en un fichero XML de configuración.

El juego puede tener varios niveles (aunque esto no lo hemos tenido en cuenta en el código de los apartados anteriores), así que en la configuración se almacenarán los datos de las diferentes pantallas (como mínimo debe existir una). En cada pantalla (nivel) hay:

- Un único jugador
- Un listado de perseguidores (que, a su vez, al menos tiene que contener un perseguidor)
- Un listado de muros (que, a su vez, puede estar vacío o tener uno o varios muros)
- Un listado de trampas (que, a su vez, puede estar vacío o tener una o varias trampas)
- Una única salida
- Además, hay que especificar el nivel de la pantalla (atributo)

Se necesita saber la posición de cada uno de estos componentes (jugador, perseguidor, muro, trampa, salida). Para facilitar el procesamiento, en vez de indicar las coordenadas como atributos de los distintos elementos, se definirá un elemento "coordenadas" que indica la posición horizontal y vertical mediante sendos atributos (el elemento coordenadas está vacío). Cada uno de los componentes del juego tiene que contener un, y sólo un, elemento coordenadas para indicar su posición.

A continuación se indica la información a almacenar en cada uno de los elementos.

Jugador: - Coordenadas (único y obligatorio)
 - Nombre (opcional)
 - URL con la imagen a mostrar (opcional)

Perseguidor: - Identificador (atributo)
 - Coordenadas (único y obligatorio)
 - Nombre (opcional)
 - URL con la imagen a mostrar (opcional)

Muro: - Coordenadas (único y obligatorio)
 - Un elemento indicando la orientación (horizontal o vertical).
 Debe aparecer obligatoriamente uno de los dos tipos, pero lógicamente sólo uno.

Trampa: - Coordenadas (único y obligatorio)
 - URL con la imagen a mostrar (opcional)

Se pide:

- Definir el DTD del fichero, de acuerdo con las especificaciones previas.
- Escribir un fichero XML de ejemplo, asociado al DTD del apartado anterior, que contenga al menos una pantalla con un componente de cada tipo.


```

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

public class Laberinto extends ... {           // ***** COMPLETAR DECLARACIÓN *****

    public static final int ARRIBA = 0;
    public static final int ABAJO = 1;
    public static final int IZDA = 2;
    public static final int DCHA = 3;

    public static int ANCHURA_VENTANA = 300;
    public static int ALTURA_VENTANA = 300;

    public static int TIEMPO_TRAMPA = 3;

    JButton[] controles;
    Casilla[][] laberinto;
    Jugador teseo;
    Perseguidor[] minotauros;

    InterfazConfiguracionXML gestorXML;

    public Laberinto(String titulo) {

        super(titulo);

        gestorXML = new GestorConfiguracionXML();
        crearLaberinto();
        crearControles();
        colocarElementosGraficos();

    }

    /**
     * Inicializa los datos del laberinto,
     * marcando adecuadamente las casillas con trampas y/o muros
     */
    public void crearLaberinto() {           // ***** COMPLETAR MÉTODO *****

        // Inicializar el tablero de casillas
        // Crear cada casilla con las características adecuadas
        // Crear los perseguidores y ponerlos en el laberinto
        // Crear el jugador y ponerlo en el laberinto

    }

    /** Prepara los botones que controlan el movimiento del jugador */
    void crearControles() {           // ***** COMPLETAR MÉTODO *****

        // Crear botones
        // Preparar oyente

    }

    /** Sitúa los componentes gráficos en la ventana */
    void colocarElementosGraficos() {           // ***** COMPLETAR MÉTODO *****

    }

}

```

```

/**
 * Visualiza un vídeo ("laberinto.mpg") en la parte central de la ventana,
 * que debe reproducirse de forma indefinida.
 *
 * Presenta un cuadro de diálogo con el mensaje adecuado según el jugador
 * haya ganado (exito==true) o perdido (exito==false)
 *
 * Cuando el jugador pulsa 'aceptar' en el cuadro de diálogo,
 * termina la aplicación.
 */
void finalizarJuego(boolean exito) {      // ***** COMPLETAR MÉTODO *****
}

public static void main(String[] args) {
    Laberinto juego = new Laberinto("Huida del laberinto");

    //manejador de evento de la ventana principal con clase adaptadora
    // para salir si se aprieta el botón de cerrar ventana (clase anónima)
    juego.addWindowListener(
        new WindowAdapter() {
            public void windowClosing(WindowEvent e)
            {System.exit(0);}
        });

    //da un tamaño a la ventana y hazla visible
    juego.setSize(ANCHURA_VENTANA, ALTURA_VENTANA);
    juego.setVisible(true);
}

// ***** COMPLETAR: programar el escuchador *****

class Casilla extends ... { // ***** COMPLETAR DECLARACIÓN *****

    final int GROSOR_MURO = 2;

    int posX;
    int posY;
    boolean esTrampa = false;
    boolean esSalida = false;
    boolean muroArriba = false;
    boolean muroAbajo = false;
    boolean muroIzda = false;
    boolean muroDcha = false;

    String imgsrc = "imgCasilla.png";

    Casilla(int f, int c, boolean esTrampa, boolean esSalida, boolean
        muroIzq, boolean muroDer, boolean muroArr, boolean muroAba) {
        super();

        this.posX = c;
        this.posY = f;

        this.esTrampa = esTrampa;
        this.esSalida = esSalida;

        this.muroIzda = muroIzq;
        this.muroDcha = muroDer;
        this.muroArriba = muroArr;
        this.muroAbajo = muroAba;
}

```

```

// Poner bordes, dependiendo de los muros
setBorder(BorderFactory.createMatteBorder
    ((muroArriba ? GROSOR_MURO : 0), // arriba
     (muroIzda ? GROSOR_MURO : 0), // izquierda
     (muroAbajo ? GROSOR_MURO : 0), // abajo
     (muroDcha ? GROSOR_MURO : 0), // derecha
     Color.BLACK // color
    ));

// Seleccionar la imagen dependiendo del tipo de casilla

if (esTrampa) imgsrc = "imgTrampa.png";
if (esSalida) imgsrc = "imgSalida.png";

// ***** COMPLETAR: dibujar la imagen en el componente *****
}

/**
 * Indica si hay muro en la casilla,
 * en la posición indicada por parámetro
 */
boolean hayMuro(int orientacion) {
    boolean hayMuro=false;
    switch (orientacion) {
        case ARRIBA:
            hayMuro = muroArriba;
            break;
        case ABAJO:
            hayMuro = muroAbajo;
            break;
        case IZDA:
            hayMuro = muroIzda;
            break;
        case DCHA:
            hayMuro = muroDcha;
            break;
    }

    return hayMuro;
}

/**
 * Elimina el personaje que había en la casilla
 * Para ello, simplemente vuelve a poner la imagen de la casilla
 * (con lo que desaparece la del personaje)
 */
void quitarPersonaje() { // ***** COMPLETAR MÉTODO *****

    // Pone la imagen de la casilla

}

/**
 * Sitúa en la casilla al personaje que se pasa como parámetro
 */
void ponerPersonaje(Personaje personaje) { // ***** COMPLETAR MÉTODO *****
    // Cambiar la imagen de la casilla por la del personaje
    // Comunicar al personaje si ha caído en una casilla activa
    // (trampa o salida), invocando a los métodos adecuados.
}
}

```

```

abstract class Personaje {
    int posX;    // columna en la que está
    int posY;    // fila en la que está
    public String imgsrc;

    Personaje(int x, int y, String imgSrc) {
        posX = x;
        posY = y;
        imgsrc = imgSrc;
    }

    int getPosX() {return posX;}
    int getPosY() {return posY;}

    boolean mover1casilla(int direccion) {

        boolean puedeMover = ! laberinto[ posY ][ posX ].hayMuro(direccion);

        if (puedeMover) {
            laberinto[ posY ][ posX ].quitarPersonaje();
            switch (direccion) {
                case ARRIBA:
                    posY--;
                    break;
                case ABAJO:
                    posY++;
                    break;
                case IZDA:
                    posX--;
                    break;
                case DCHA:
                    posX++;
                    break;
            }
            laberinto[ posY ][ posX ].ponerPersonaje(this);
        }

        return puedeMover;
    }

    /** Caer en trampa tiene distintos efectos según el tipo de personaje */
    abstract void caeEnTrampa();

    /** Sólo hace algo si es el jugador */
    void caeEnSalida() {};
}

class Jugador extends Personaje {
    Jugador(int fila, int columna, String rutaImg) {
        super(fila, columna, rutaImg);
    }

    void caeEnTrampa() {
        // Fin del juego: ha caído en una trampa
        finalizarJuego(false);
    }

    void caeEnSalida() {
        // Fin de juego: gana
        finalizarJuego(true);
    }
}

```

```

class Perseguidor extends Personaje {

    int tiempoEspera;    // número de turnos sin mover (cuando está en trampa)

    Perseguidor(int posX, int posY, String rutaImg) {
        super(posX, posY, rutaImg);
        tiempoEspera = 0;
    }

    /**
     * Mueve al perseguidor, siguiendo las siguientes normas:
     *
     * 1º intenta acercarse al jugador en horizontal
     * Si no puede (porque hay un muro) o ya está en la misma columna que el
     * jugador, intenta moverse en vertical.
     * Si no puede moverse tampoco en vertical (porque haya un muro),
     * pasa su turno.
     *
     * No es "inteligente", es decir, no intenta evitar las trampas,
     * ni los muros...
     *
     * El método debe comprobar si se ha alcanzado al protagonista,
     * en cuyo caso se invoca al método finalizarJuego.
     *
     * Si está en una trampa, pasa el turno sin mover,
     * simplemente disminuye el tiempo de espera
     */
    void mover() {

        // ***** COMPLETAR *****

    }

    void caeEnTrampa(){
        tiempoEspera = TIEMPO_TRAMPA;
    }

}

/**
 * Clase que permite leer la configuración del laberinto de un fichero XML
 */
class GestorConfiguracionXML implements InterfazConfiguracionXML {

    // Supón que esta clase está implementada

}
}

```

```

interface InterfazConfiguracionXML {

    /** Devuelve el número de columnas del laberinto */
    int getAnchuraLaberinto();

    /** Devuelve el número de filas del laberinto */
    int getAlturaLaberinto();

    /** Indica si la casilla situada en la fila y columna indicadas es trampa */
    boolean esTrampa(int fila, int columna);
    /** Indica si la casilla situada en la fila y columna indicadas es salida */
    boolean esSalida(int fila, int columna);

    /**
     * Indica si la casilla situada en la fila y columna indicadas
     * tiene muro izquierdo
     */
    boolean hayMuroIzq(int fila, int columna);

    /**
     * Indica si la casilla situada en la fila y columna indicadas
     * tiene muro derecho
     */
    boolean hayMuroDer(int fila, int columna);

    /**
     * Indica si la casilla situada en la fila y columna indicadas
     * tiene muro arriba
     */
    boolean hayMuroArr(int fila, int columna);

    /**
     * Indica si la casilla situada en la fila y columna indicadas
     * tiene muro abajo
     */
    boolean hayMuroAba(int fila, int columna);

    /** Devuelve el número de perseguidores */
    int getNumeroPerseguidores();

    /** Devuelve la posición horizontal (columna) inicial del perseguidor i */
    int getPosicionXPerseguidor(int i);

    /** Devuelve la posición vertical (fila) inicial del perseguidor i */
    int getPosicionYPerseguidor(int i);

    /** Devuelve el nombre de fichero con la imagen del perseguidor i */
    String getRutaImagenPerseguidor(int i);

    /** Devuelve la posición horizontal (columna) inicial del jugador */
    int getPosicionXJugador();

    /** Devuelve la posición vertical (fila) inicial del jugador */
    int getPosicionYJugador();

    /** Devuelve el nombre de fichero con la imagen del jugador */
    String getRutaImagenJugador();

}

```