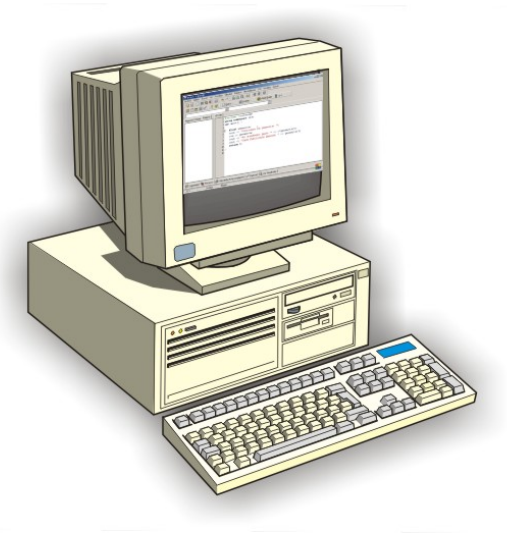

Programación J2ME

Construcción del juego de la serpiente

Septiembre de 2006

DECSAI
Departamento de Ciencias de la Computación e I.A.
Universidad de Granada

Curso de Formación Continua de Programación de dispositivos móviles con Java (4ª edición)

Septiembre de 2006

Índice

Figura 1: Ejecución del programa con ktoolbar

1. Introducción

El objetivo de este ejercicio es construir un sencillo juego conocido como *juego de la serpiente* (*snake* en inglés). El aspecto del juego al ejecutarlo en el emulador de *ktoolbar* sería el de la figura 2.

Este juego consiste en controlar el movimiento de la serpiente mediante las flechas de dirección del teclado del dispositivo móvil, intentando comer los ratones que se mueven de forma aleatoria. Cada vez que se come un ratón aumenta la longitud de la serpiente y obtenemos un punto más. El ratón aparece entonces nuevamente en una posición aleatoria de la pantalla. La serpiente muere cuando choca contra sí misma o contra una de las paredes.

2. Ejecución del programa con ktoolbar

Podéis comprobar vosotros mismos el funcionamiento de este programa mediante *ktoolbar* tomando el fichero `../serpienteMiddletTuring.tar.gz` y descompriéndolo en el directorio `$HOME/WTK2.2/apps`. En li-

nux, para descompimir el fichero `../serpienteMiddletTuring.tar.gz` en `$HOME/WTK2.2/apps`, cópialo primero a algún directorio de tu cuenta, por ejemplo al mismo directorio `$HOME/WTK2.2/apps`, cámbiate al directorio `$HOME/WTK2.2/apps` y ejecuta el comando:

```
tar zxvf serpienteMiddlet.tar.gz
```

Esto hará que se cree el directorio `Serpiente` dentro de `apps`. Ejecuta ahora *ktoolbar* y abre el proyecto *Serpiente*. Ejecuta el proyecto seleccionando el botón **Run**. El emulador que aparece tiene el aspecto del de la figura 2.

Si miramos la estructura de directorios generada al descomprimir `serpienteMiddletTuring.tar.gz` podemos ver que bajo el subdirectorio `src` no se encuentran los ficheros `.java` del código fuente. La razón es que esto es precisamente lo que se os pide a vosotros que hagáis, el código fuente de esta aplicación.

3. Ejecución de la versión AWT del programa

El interfaz gráfico de esta aplicación necesita de las clases de MIDP que se estudiarán en este curso más adelante (`http://leo.ugr.es/J2ME/MIDP/index2.htm`). Por ahora, comenzaremos a construir las clases que no necesitan de las clases de MIDP.

Para que podáis ir comprobando que las clases las estáis construyendo correctamente, se proporciona la versión *AWT* de este mismo programa, en el fichero comprimido `../serpienteAWT.tar.gz`. *AWT* es un conjunto de clases que permiten construir el interfaz gráfico de un programa java para ejecutarlo en un ordenador normal (no en un dispositivo móvil) con *jdk*. En las aulas de ordenadores como se ya se habrá comentado, está instalada la versión 1.5 del jdk en el directorio `/usr/local/jdk1.5.0_06` Los ejecutables (`java` y `javac`) necesarios para ejecutar y compilar programas jdk ya deben estar añadidos a vuestro *path*, por lo que no habría que hacer nada especial para poder usarlos.

En este caso, el fichero `../serpienteAWT.tar.gz` sí que contiene el código fuente de algunas clases, pero no de todas. Contiene el código fuente de las clases correspondientes al interfaz gráfico para AWT, concretamente de las clases *MiFrame.java* (que representa la ventana donde aparece la aplicación), *MiCanvas.java* (el panel contenido en la anterior ventana, donde se dibuja el tablero de juego) y *Main.java* (que básicamente contiene la función *main()* que es por donde comienzan a ejecutarse los programas jdk). Para el resto de clases sólo se proporciona el fichero *.class* para que podáis ejecutar el programa, pero no se proporciona el fichero *.java*, ya que según se ha diseñado el programa, la versión J2ME y la versión para JDK de estos clases puede ser exactamente la misma.

Descargad el fichero `../serpienteAWT.tar.gz` y descomprimidlo en algún lugar de vuestra cuenta. Por ejemplo podéis descargarlo en el directorio *SerpienteAWT*, cambiaros a él y ejecutar el comando:

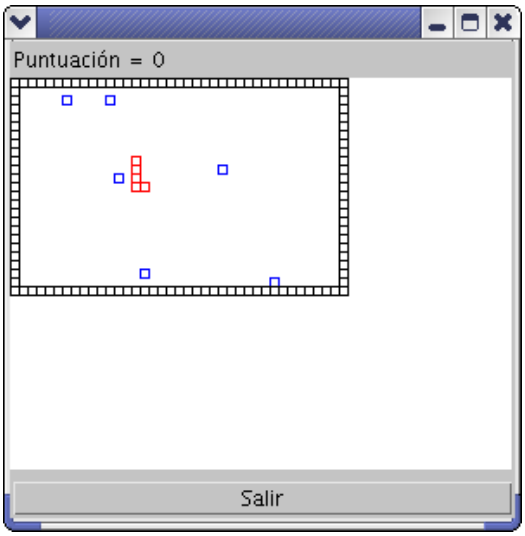

Figura 2: Ejecución de la versión AWT del programa

```
tar zxvf serpienteAWT.tar.gz
```

Esto hará que se cree el directorio *serpiente* dentro de *SerpienteAWT*, que contendrá los ficheros .java mencionados más arriba junto con todos los ficheros .class de la aplicación. También contiene un subdirectorio *javadocs*[1] que corresponde a la documentación estilo *javadoc* de las clases de esta aplicación. Los anteriores ficheros también están disponibles accediendo a la dirección web:
http://leo.ugr.es/J2ME/APPS/Serpiente/SerpienteAWT.

Para ejecutar esta versión AWT del programa de la serpiente podéis ejecutar el siguiente comando (suponiendo que estamos colocados en el directorio padre del directorio *serpiente*).

```
java serpiente.Main
```

La documentación javadoc la podéis consultar con cualquier navegador web, tal como el *firefox*, abriendo el fichero *index.html* que se encuentra dentro del directorio `serpiente/javadocs`. Esta misma documentación está también disponible en:
http://leo.ugr.es/J2ME/APPS/Serpiente/SerpienteAWT/javadocs

4. Construcción de las clases básicas del programa

La idea en esta sección es que vayamos construyendo los ficheros *.java* que nos faltan (Punto.java, Rejilla.java, Raton.java, Ratones.java, Serpiente.java y Mueve.java) en la versión AWT del programa de la serpiente

[1] Puedes aprender cómo documentar un programa Java con estilo javadoc en la dirección http://java.sun.com/j2se/javadoc/index.jsp

que se ha comentado en la sección anterior. Cuando hagamos la versión J2ME de este programa, podremos reutilizar estos ficheros sin tener que modificar nada. En la versión J2ME podemos reutilizar todos los ficheros .java del programa, excepto Main.java, MiCanvas.java y MiFrame.java. Para la versión J2ME sólo nos faltará hacer el interfaz gráfico de la aplicación usando las clases de MIDP.

El orden en el que se recomienda implementar las anteriores clases es el siguiente:

1. Punto.java
2. Rejilla.java
3. Raton.java
4. Ratones.java
5. Serpiente.java
6. Mueve.java

Comenzaremos implementando por tanto la clase Punto.java. Para ello consulta la documentación html de javadoc para esta clase disponible en:

`http://leo.ugr.es/J2ME/APPS/Serpiente/SerpienteAWT/javadocs`

Esta documentación nos permite saber qué datos miembro, constructores y métodos necesitamos incluir en cada clase.

4.1. Clase Punto

Dentro del directorio `serpiente` abre algún editor de texto (Por ejemplo *kwrite*) para construir la clase Punto.java:

```
kwrite Punto.java
```

Si miramos la documentación html de la clase Punto, podemos ver que es una clase muy simple. Sólo hay que añadir dos datos miembro privados (`private`) de tipo `int` y un constructor `Punto(int coorx, int coory)` que se encarga de inicializar las coordenada `x` e `y`. Por tanto esta clase podría quedar de la siguiente forma:

```
package serpiente;

/**
 * Clase que representa unas coordenadas x e y enteras
 */

public class Punto{
  int x,y;
  Punto(int coorx, int coory){
    x=coorx;
    y=coory;
  }
}
```

Fíjate que hemos incluido la sentencia `package serpiente;` al principio del fichero para indicar que la clase pertenece al paquete `serpiente`. Esto lo haremos también con el resto de clases.

Compilemos ahora esta clase para ver si el programa AWT sigue funcionando. Al hacerlo se sustituirá el fichero .class proporcionado con el asociado a la clase Punto que acabamos de construir. Para compilar el fichero nos colocamos en el directorio padre de `serpiente` y ejecutamos:

```
javac serpiente/Punto.java
```

Si todo ha ido bien tendremos un nuevo fichero Punto.class en el directorio `serpiente`. Vuelve a ejecutar el programa AWT para ver si sigue funcionando bien:

```
java serpiente.Main
```

4.2. Clase Rejilla

Como indica la documentación html de javadoc para esta clase, representa una rejilla (matriz bidimensional) con una determinada Anchura y Altura, en la que cada celda puede estar VACIA, contener un trozo de SERPIENTE, un RATON, o un BLOQUE (muro). La matriz bidimensional se almacena en el dato miembro:

```
private int[][] celdas;
```

En esta matriz el primer índice se usa para la anchura (para las coordenadas x) y el segundo índice para la altura (para las coordenadas y). Otros datos miembro que contiene esta clase son la anchura y altura que en mi caso están inicializadas con los valores 39 y 25, pero que tú podrías inicializar con otros valores distintos:

```
private int anchura=39;
private int altura=25;
```

Además debemos incluir los siguientes datos `static final`:

```
static final int TIPO_MASCARA = 0x00FF0000; // Tipos de celda de la rejilla
static final int VACIA        = 0x00000000;
static final int BLOQUE       = 0x00010000;
static final int SERPIENTE    = 0x00020000;
static final int RATON        = 0x00030000;
```

Las constantes VACIA, BLOQUE, SERPIENTE y RATON son los distintos valores que pueden incluirse en cada una de las celdas de la rejilla. O sea, si por ejemplo en un momento dado, en la posición 5,7 tenemos el valor SERPIENTE, entonces quiere decir que en esa posición hay un trozo de serpiente.

Usa un editor de texto (*kwrite*) para construir el fichero Rejilla.java, e incluye los anteriores datos miembro:

```
package serpiente;

/**
 * Esta clase representa una rejilla con una determinada Anchura
 * y Altura, en la que cada celda puede estar VACIA, contener
 * un trozo de SERPIENTE, un RATON, o un BLOQUE (muro)
 */
public class Rejilla{
  static final int TIPO_MASCARA = 0x00FF0000; // Tipos de celda de la rejilla
  static final int VACIA        = 0x00000000;
  static final int BLOQUE       = 0x00010000;
  static final int SERPIENTE    = 0x00020000;
  static final int RATON        = 0x00030000;

  private int anchura=39;
  private int altura=25;

  private int[][] celdas;
}
```

Añade ahora el constructor y el resto de métodos de esta clase teniendo en cuenta su documentación javadoc. Cuando acabes, compila el fichero Rejilla.java de la misma forma que en la sección anterior con Punto.java, y ejecuta de nuevo el programa para ver si sigue funcionando. Sólo es necesario implementar los métodos públicos. Los métodos privados no es necesario implementarlos.

4.3. Clase Raton

Esta clase sirve para almacenar un ratón. Un Raton se define mediante unas coordenadas x e y, y una dirección de movimiento. En este caso debemos incluir en la clase un dato miembro `xy` de la clase Punto, un dato `direccion` de tipo `int` para indicar la dirección en que se mueve actualmente el ratón. También tenemos que incluir los datos `static final` `NINGUNA`, `IZQUIERDA`, `DERECHA`, `ARRIBA`, `ABAJO` y `DIRECCION_MASCARA`. Finalmente es necesario el dato miembro `rand` de la clase `Random` que se usará para generar números aleatorios, para así generar aleatoriamente la dirección de movimiento del ratón, y la posición x,y inicial donde debe aparecer. La clase quedaría así:

```
package serpiente;

import java.util.Random;

/**
 * Esta clase sirve para almacenar un ratón. Un Raton se define
 * mediante unas coordenadas x e y, y una dirección de movimiento.
 */
public class Raton{
  static final int DIRECCION_MASCARA = 0x0000FF00;
  static final int NINGUNA           = 0x00000000;
  static final int IZQUIERDA         = 0x00000100;
  static final int DERECHA           = 0x00000200;
  static final int ARRIBA            = 0x00000300;
  static final int ABAJO             = 0x00000400;

  private Punto xy;
  private int direccion;
  static private Random rand=new Random();
}
```

Añade el constructor y el resto de los métodos teniendo en cuenta la documentación javadoc para esta clase. Compila y ejecuta de nuevo el programa.

4.4. Clase Ratones

Esta clase sirve para almacenar un conjunto de objetos de la clase Raton. El número de Ratones es el definido por la constante `NUMERO_RATONES`. Los Ratones los consideraremos numerados desde 0 hasta `NUMERO_RATONES-1`. La estructura de datos que podemos elegir para almacenar el conjunto de ratones es un array, aunque es perfectamente válido usar otra estructura de datos distinto. Si usamos un array entonces el dato miembro `ratones` tendría la forma:

```
private Raton[] ratones;
```

Además tenemos que añadir el dato miembro `rejilla` que simplemente será una referencia a la `Rejilla` del juego pues la necesitaremos en algunos métodos. También hay que incluir el dato miembro `static final NUMERO_RATONES` que indica el número de ratones que hay en el juego. En mi caso está inicializado al valor 6, aunque tú puedes elegir el valor que quieras.

La clase quedaría así:

```
package serpiente;

/**
 * Esta clase sirve para almacenar un conjunto de objetos
 * de la clase Raton. El número de Ratones es el definido
 * por la constante NUMERO_RATONES. Los Ratones los consideraremos
 * numerados desde 0 hasta NUMERO_RATONES-1.
 */

public class Ratones{
  public static final int NUMERO_RATONES=6;
  private Raton[] ratones;
  private Rejilla rejilla;
}
```

Añade ahora el constructor y el resto de los métodos. Compila y ejecuta.

4.5. Clase Serpiente

Esta clase representa la Serpiente. Una Serpiente está formada por un número variable de coordenadas x e y, que se guardan en un objeto de la clase Vector. Cada elemento en este Vector, será un objeto de la clase Punto, que nos indica la coordenada x e y dónde está colocado ese trozo de Serpiente. El número de celdas puede ir aumentando y disminuyendo en tiempo de ejecución del programa. La Serpiente también necesita de una dirección de movimiento (direccion) y una referencia (cabezaPtr) a la posición en el Vector de celdas dónde está la cabeza.

El Vector actua como un vector circular. Puede pensarse en la serpiente como un conjunto de celdas, cuya primera posición (cabeza de la serpiente) es el elemento del Vector apuntado por cabezaPtr, la segunda posición sería la siguiente y así sucesivamente. La siguiente posición al último elemento en el Vector sería el primer elemento del Vector. Esta estructura de datos para la serpiente, facilitamente mucho la implementación del método `mueveSerpiente()`. La implementación de este método podría ser:

```
public int mueveSerpiente(Ratones ratones){
  int c,d,i,j;
  Punto pt;

  // Calculamos posicion nueva cabeza
  pt=(Punto)celdas.elementAt(cabezaPtr);
  i=pt.x;
  j=pt.y;
  switch(direccion){
  case IZQUIERDA:
    i--;
    break;
  case DERECHA:
    i++;
    break;
  case ARRIBA:
    j--;
    break;
  case ABAJO:
    j++;
    break;
  default:
    break;
  }

  // Comprobamos si nueva cabeza chocará con algo
  c=rejilla.getTipoCelda(i,j);
  if(c == Rejilla.BLOQUE){
    System.out.println("Te has chocado contra muro ");
    return CHOCA_BLOQUE;
  }
  else if(c== Rejilla.SERPIENTE){
    System.out.println("Te has chocado contra muro ");
    return CHOCA_SERPIENTE;
  }
  else if(c == rejilla.RATON){ // La serpiente crece de tamaño
    System.out.println("Te has comido un raton");
    // Insertar nuevo punto para cabeza
    pt=new Punto(i,j);
    celdas.insertElementAt(pt,cabezaPtr);
    rejilla.assignTipoCelda(i,j,Rejilla.SERPIENTE);
    int n=ratones.getNumeroRaton(i,j);
    if(n!=-1){
      ratones.initRaton(n);
    }
    return COME_RATON;
  }
  else{ // Si no choca con nada
    int colaPtr;
    // En la rejilla ponemos en la nueva celda de cabeza un trozo de SERPIENTE
    cabezaPtr--;
    rejilla.assignTipoCelda(i,j,Rejilla.SERPIENTE);
    if(cabezaPtr < 0 )
      cabezaPtr=celdas.size()-1;

    // En la rejilla dejamos vacia la posicion de la antigua cola (nueva cabeza)
    pt=(Punto)celdas.elementAt(cabezaPtr);
    rejilla.assignTipoCelda(pt.x,pt.y,Rejilla.VACIA);
    // Colocamos en lista de celdas de serpiente las coordenadas de nueva cabeza
    ((Punto)celdas.elementAt(cabezaPtr)).x=i;
    ((Punto)celdas.elementAt(cabezaPtr)).y=j;
```

```
      return HABIA_NADA;
    }
  }
```

La clase también define los datos `static final` NINGUNA, IZQUIERDA, DERECHA, ARRIBA, ABAJO y DIRECCION_MASCARA necesarios para definir la dirección actual de movimiento de la serpiente. Tenemos también los datos miembro HABIA_NADA, CHOCA_BLOQUE, CHOCA_SERPIENTE y COME_RATON que corresponden a las distintas situaciones que se pueden presentar cuando la cabeza de la serpiente avanza una casilla. Finalmente la clase contiene el dato miembro `rejilla` que es una referencia a la Rejilla del juego, necesaria en algunos métodos de la clase. La clase quedaría de la siguiente forma:

```
package serpiente;

import java.util.Vector;

/**
 * Esta clase representa la Serpiente. Una Serpiente está formada
 * por un número variable de celdas adyacentes, que se guardan en
 * un Vector. El número de celdas puede aumentar si se desea.
 * La Serpiente también necesita de una dirección de movimiento (direccion) y
 * un puntero (cabezaPtr) a la posición en el Vector de celdas dónde
 * está la cabeza.
 */


public class Serpiente{
  static final int DIRECCION_MASCARA = 0x0000FF00;
  static final int NINGUNA           = 0x00000000;
  static final int IZQUIERDA         = 0x00000100;
  static final int DERECHA           = 0x00000200;
  static final int ARRIBA            = 0x00000300;
  static final int ABAJO             = 0x00000400;

  static final int HABIA_NADA      = 0; // Posibles situaciones al mover cabeza
  static final int CHOCA_BLOQUE    = 1;
  static final int CHOCA_SERPIENTE = 2;
  static final int COME_RATON      = 3;

  private Vector celdas;
  private Rejilla rejilla;
  private int cabezaPtr;
  private int direccion;
}
```

Añade tú el constructor y el resto de los métodos teniendo en cuenta los comentarios javadoc. Compila y ejecuta la el programa.

4.6. Clase Mueve

Esta clase implementa una hebra que hace que se muevan continuamente la serpiente y los ratones. La hebra se encarga también de ir refrescando la pantalla dónde se dibuja todo, y los puntos acumulados. Además controla si la serpiente choca contra un muro o contra sí misma, para comenzar el juego de nuevo. Cuando la serpiente come un Raton aumenta su longitud en una celda. En este caso los datos miembro pueden verse a continuación.

```
package serpiente;
import java.awt.Canvas;
import java.awt.Label;

/**
  * Esta clase implementa una hebra que hace que se muevan continuamente la serpiente
  * y los ratones. La hebra se encarga también de ir refrescando la pantalla
  * dónde se dibuja todo, y los puntos acumulados. Además controla si
  * la serpiente choca contra un muro o contra sí misma, para comenzar
  * el juego de nuevo. Cuando la serpiente come un Raton aumenta su longitud
  * en una celda.
  */
public class Mueve implements Runnable{
  private Serpiente serpiente;
  private Ratones ratones;
  private Rejilla rejilla;
  private int delay;
  private boolean continuar=true;
  private boolean suspendFlag=true;
  private Canvas canvas=null; // donde se pinta todo
  private Label labelPuntuacion;
  private int puntuacion;
```

El dato `serpiente` es una referencia a la Serpiente del juego. `ratones` es una referencia al vector de ratones del juego. `rejilla` es una referencia a la Rejilla del juego. `delay` es un entero que representa el número de milisegundos que se detiene el juego entre cada movimiento de la serpiente y ratones. Cuanto menor sea, a mayor velocidad se moverán. Puede verse que se inicializa con un parámetro de los constructores. Puede verse que el objeto de la clase Mueve es creado en la clase Main, pasando el valor 200 en este parámetro. `continuar` y `suspendFlag` son las variables que se usan en el método `run()` para controlar la hebra en la forma que se ha explicado en las trasparencias de introducción a Java. `canvas` es una referencia al objeto Canvas (panel donde se dibuja la serpiente, ratones, etc). `labelPuntuacion` es una referencia a un objeto de la clase Label, una clase de AWT que permite mostrar un texto, y que es usado para mostrar la puntuación obtenida hasta el momento. `puntuacion` es el número de puntos conseguidos hasta el momento.

Como ayuda a continuación mostramos los constructores y los métodos `run()` y `suspender()` de esta clase:

```
  /**
   * Constructor de la clase, que inicializa las referencias utilizadas por
   * la hebra a Rejilla, Serpiente y Ratones, establece el retardo en milisegundos
   * entre movimiento y movimiento de Serpiente y Ratones, y comienza a ejecutar
   * la hebra.
   */
  Mueve(Rejilla rej,Serpiente serp,Ratones rat,int retardo){
    rejilla=rej;
    serpiente=serp;
    ratones=rat;
    delay=retardo;
    Thread t=new Thread(this);
    t.start();
  }

  /**
   * Constructor de la clase, que inicializa las referencias utilizadas por
   * la hebra a Rejilla, Serpiente y Ratones, establece el retardo en milisegundos
   * entre movimiento y movimiento de Serpiente y Ratones, y comienza a ejecutar
   * la hebra. Además inicializa la referencia al Canvas dónde se dibuja
   * Serpiente y Ratones, y la refencia al Label dónde aparece la puntuación.
   */
```

```
Mueve(Rejilla rej,Serpiente serp, Ratones rat,int retardo, Canvas c,Label label){
  rejilla=rej;
  canvas=c;
  labelPuntuacion=label;
  serpiente=serp;
  ratones=rat;
  delay=retardo;
  puntuacion=0;
  Thread t=new Thread(this);
  t.start();
}

/**
 * Código que constituye las sentencias de la hebra. En este caso, se encarga
 * de hacer que se muevan continuamente la Serpiente
 * y los Ratones. La hebra se encarga también de ir refrescando la pantalla
 * dónde se dibuja todo, y los puntos acumulados. Además controla si
 * la Serpiente choca contra un muro o contra sí misma, para comenzar
 * el juego de nuevo. Cuando la Serpiente come un Raton aumenta su longitud
 * en una celda.
 */
public void run(){
  try{
    while(continuar){
      synchronized(this){
        while(suspendFlag){
          wait();
        }
      }
      Thread.sleep(delay);
      int m=serpiente.mueveSerpiente(ratones);
      ratones.mueveRatones();
      if(canvas!=null)
        canvas.repaint();

      if((m==Serpiente.CHOCA_BLOQUE)||(m==Serpiente.CHOCA_SERPIENTE)){
        suspender();
        inicializaJuego();
        canvas.repaint();
        labelPuntuacion.setText("Puntuación = "+puntuacion);
        reanudar();

      }
      else if(m==Serpiente.COME_RATON){
        puntuacion++;
        labelPuntuacion.setText("Puntuación = "+puntuacion);
      }
    }// end while(continuar)
  }
  catch(InterruptedException e){
    System.out.println("Hilo MueveSerpiente interrumpido");
  }
}

/**
  * Detiene momentaneamente la ejecución de la hebra, haciendo que la
  * Serpiente y Ratones queden parados.
  */
synchronized public void suspender(){
  suspendFlag=true;
}
```

Implementa el resto de los métodos de la clase teniendo en cuenta las especificaciones proporcionadas en la documentación javadoc.

5. Construcción del interfaz gráfico MIDP

Haciendo uso de ktoolbar o bien de netbeans construye el resto de clases necesarias para construir la versión J2ME de esta aplicación.