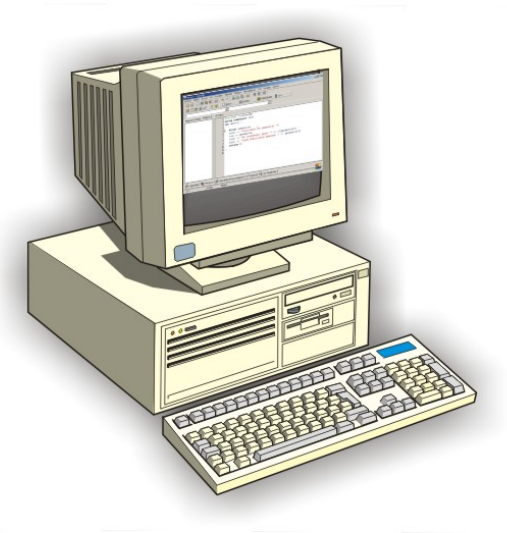


Práctica 8

Práctica evaluable*: Juego del comecocos en Java*

Mayo de 2015

DECSAI

Complementos de Programación

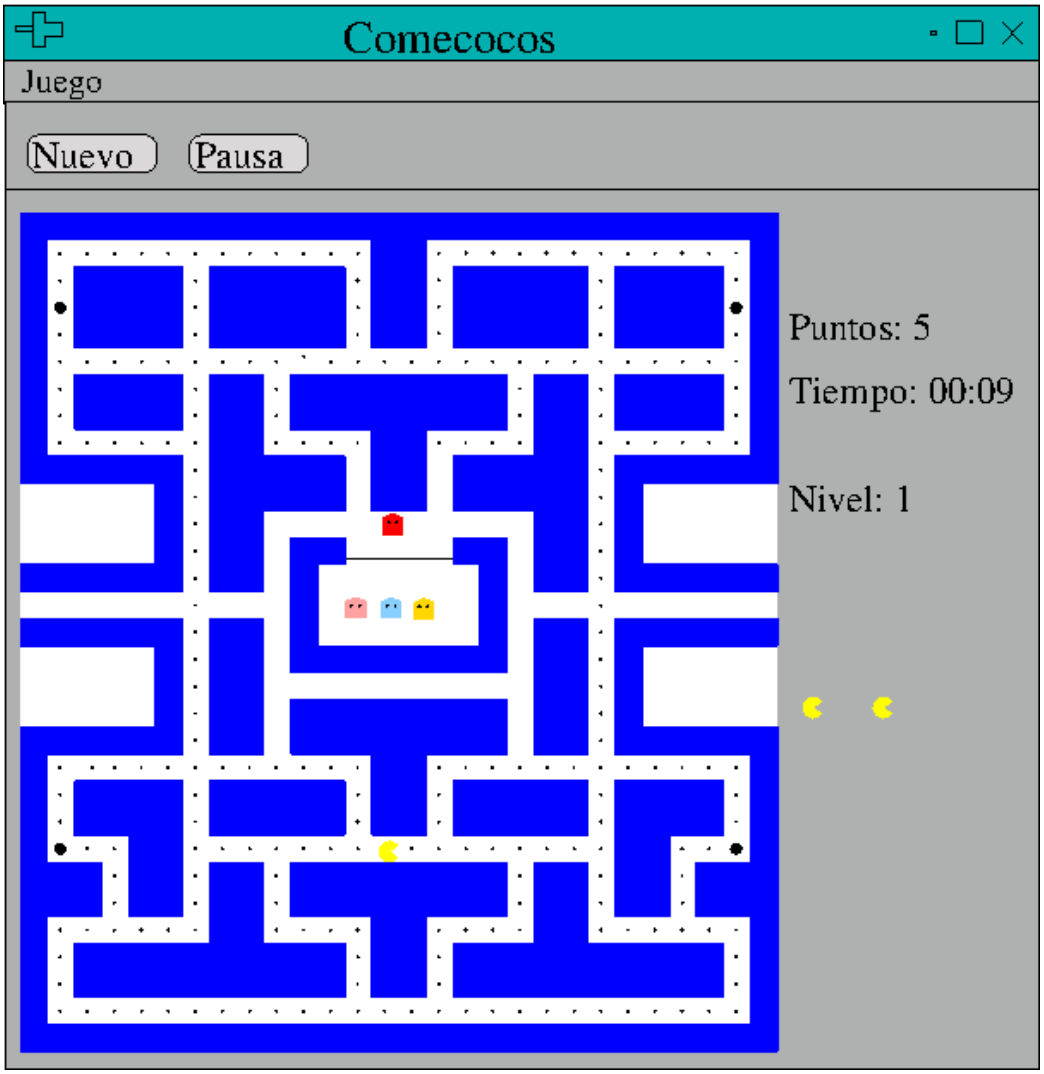
Curso 2014/2015

Contents

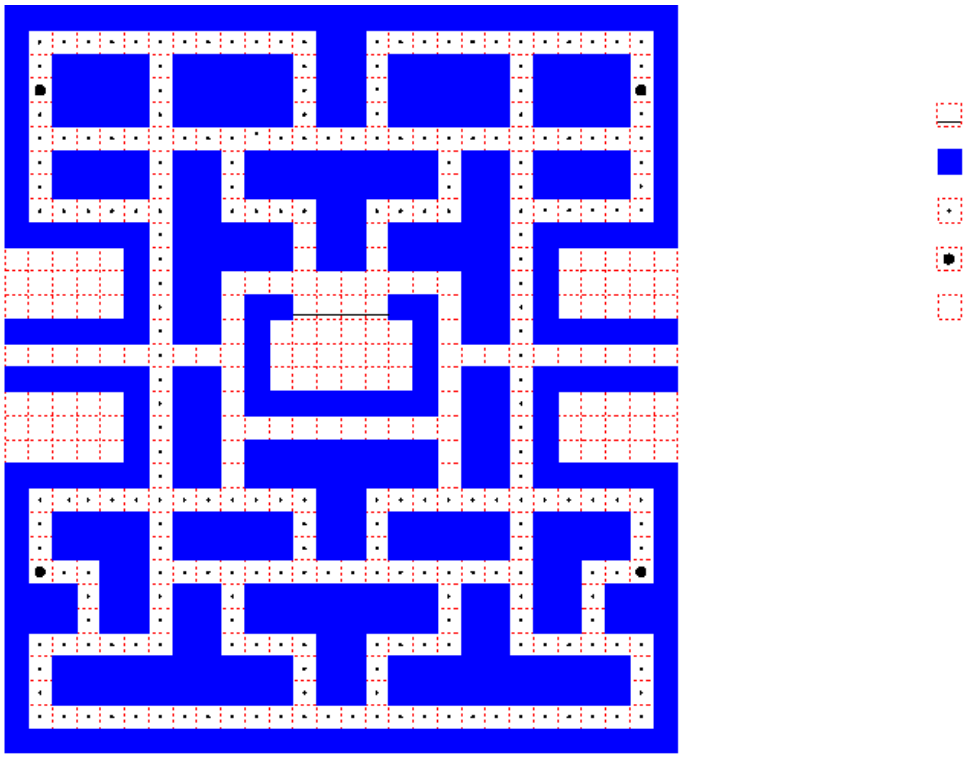
1 Descripción

La práctica consiste en la implementación en Java utilizando el kit de desarrollo jdk (se recomienda utilizar la versión jdk 1.8) de un programa para jugar a una versión simplificada del juego del **Comecocos**. En este juego el usuario debe controlar un *comecocos* (en amarillo en la figura de más abajo), mediante las cuatro teclas de dirección del teclado. El objetivo del juego es moverse por el laberinto comiendo todos los puntos pequeños y grandes, sin que sea alcanzado por ninguno de los cuatro fantasmas. Los fantasmas se mueven de forma más o menos aleatoria por el laberinto intentando alcanzar al comecocos. Cada vez que el *comecocos* come un punto pequeño se consiguen 10 puntos. Al comer uno de los puntos grandes se consiguen 50 puntos, y además los fantasmas pasan a estado *comestible*, o sea que ahora es el comecocos el que puede comer a los fantasmas durante un pequeño intervalo de tiempo. En ese caso, al comer el primer fantasma se consiguen 200 puntos, 400 puntos con el segundo, 800 puntos con el tercero y 1600 puntos con el cuarto.

El interfaz del programa puede realizarse con `AWT` o bien con `Swing`, aunque se recomienda el uso de `Swing`. Para construir el programa puede utilizarse la herramienta visual **Netbeans**, pero hay que asegurarse que el programa funciona luego con `jdk`.



Para modelizar el laberinto, puede usarse una rejilla bidimensional de tamaño 28 de ancho por 31 de alto. Cada celda de la rejilla puede contener un trozo de bloque (muro), un punto pequeño, un punto grande, espacio vacío o un trozo de puerta (se usa como puerta de la caja donde están los fantasmas). En la siguiente figura aparecen marcadas en rojo las celdas de la rejilla. Al lado derecho del laberinto aparecen los cinco tipos de celdas necesarias para contruir el laberinto.



.

Para definir el estado inicial de la rejilla debemos utilizar un array de `Strings` de la siguiente forma:

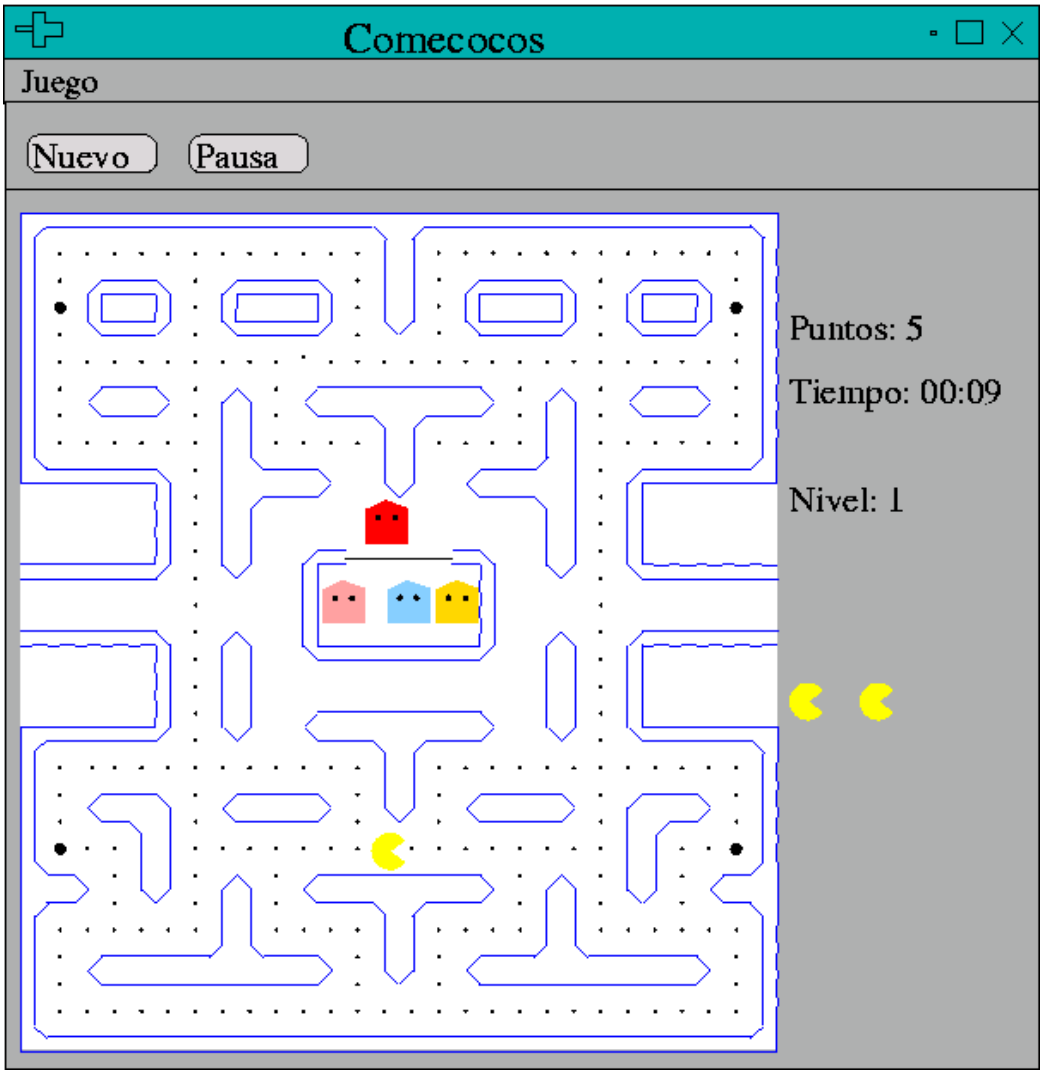
```
String rejilla[]={
"BBBBBBBBBBBBBBBBBBBBBBBBBBBB",
"B............BB............B",
"B.BBBB.BBBBB.BB.BBBBB.BBBB.B",
"BoBBBB.BBBBB.BB.BBBBB.BBBBoB",
"B.BBBB.BBBBB.BB.BBBBB.BBBB.B",
"B..........................B",
...
};
```

El anterior array codifica las seis primeras filas del laberinto, usando la siguiente codificación:

- Bloque: `B`
- Punto pequeño: `.`
- Punto grande: `o`

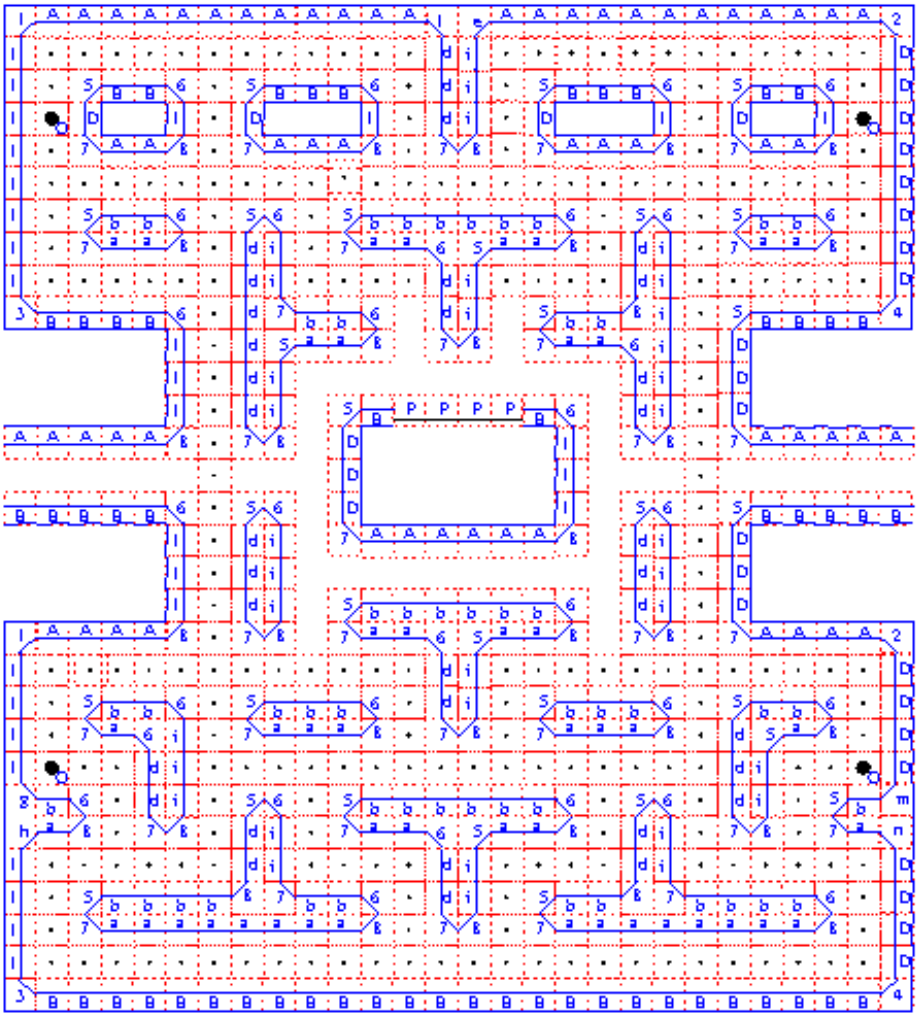
Podemos añadir una clase (por ejemplo llamada `Laberinto`) que construya el laberinto a partir del anterior array. Esta clase sería similar a la clase `Rejilla` que se ha usado en otras prácticas.

Podríamos mejorar el aspecto del laberinto si utilizamos distintos tipos de celdas para los bloques, según muestra la siguiente figura:



Mostramos de nuevo la rejilla, junto con los tipos de celdas necesarias para este nuevo laberinto detallado. En este caso las seis primeras filas de la rejilla podrían codificarse con el siguiente vector de `String`:

```
String rejilla[]={
"1AAAAAAAAAAAAfeAAAAAAAAAAAA2",
"I............di............D",
"I.5BB6.5BBB6.di.5BBB6.5BB6.D",
"IoD  I.D   I.di.D   I.D  IoD",
"I.7AA8.7AAA8.78.7AAA8.7AA8.D",
"I.........................D",
...
};
```



2 Requerimientos mínimos

Los requerimientos mínimos que debe cumplir la práctica son los siguientes:

1. El mapa del laberinto debe definirse a partir de un array de Strings según se ha indicado anteriormente.
2. Debe haber alguna forma de comenzar un nuevo juego y salir del programa (opciones de un menú o bien botones).
3. Sólo es obligatorio que aparezca el comecocos. Los fantasmas no son obligatorios.

4. No es obligatorio que aparezcan en el tablero los puntos pequeños y puntos grandes.
5. El comecocos puede dibujarse con un círculo relleno en color amarillo.
6. Como opción básica, el movimiento del comecocos será celda a celda.
7. Una vez que el comecocos comience a moverse, seguirá haciéndolo en la dirección actual sin necesidad de pulsar ninguna tecla, mientras no encuentre un obstáculo.
8. El movimiento del comecocos se hará a través de una hebra, en la que en un bucle infinito se calculará la nueva posición del comecocos, y se redibujará en la nueva posición, durmiendo la hebra a continuación durante un pequeño instante de tiempo.
9. Deben controlarse las teclas de cursor (izquierda, derecha, arriba y abajo) del teclado para cambiar la dirección de movimiento del comecocos.
10. En la opción básica, el laberinto se dibujará en la primera forma (la más sencilla).

3 Mejoras propuestas

Opcionalmente se pueden incluir en el programa otras **mejoras**, explicándolas en la documentación del programa. Estas mejoras no son obligatorias y se usarán para subir nota. Por ejemplo, algunas mejoras podrían ser:

1. Se pueden incluir los puntos pequeños y grandes en el tablero de forma que:
   - Cuando el comecocos pasa por encima de un punto pequeño o grande, éste desaparece y se incrementa la puntuación obtenida. Se consiguen 10 puntos al comer un punto pequeño y 50 al comer uno grande.
   - Los puntos conseguidos hasta el momento deben ser visibles en alguna parte del juego (10 puntos por punto pequeño y 50 puntos por punto grande).
   - El juego acaba cuando el comecocos come todos los puntos del laberinto inicial.
2. Dibujar el laberinto según la forma detallada mostrada u otra similar.
3. Incluir alguna forma de detener el juego y reanudarlo posteriormente. Por ejemplo, podría ser un botón del interfaz y también con la pulsación de la barra espaciadora.

4. Para que el movimiento del comecocos sea más continuo, en lugar de hacerlo celda a celda, podrían hacerse 4 pasos entre cada celda.

5. Se pueden utilizar diferentes aspectos para el comecocos según sea su dirección de movimiento.
   Además entre celda y celda, si implementamos la opción de más arriba, podríamos utilizar un comecocos con distinta apertura para la boca.

6. Hacer que aparezcan los cuatro fantasmas. Estos irían saliendo poco a poco de la caja dónde se encuentran encerrados inicialmente. La velocidad de los fantasmas debe ser algo menor a la del comecocos. Cada fantasma sería animado creando una hebra que se encargaría de mover el fantasma continuamente.

   - Cuando el comecocos come uno de los puntos grandes los fantasmas pasan a estado *comestible* durante un pequeño intervalo de tiempo. En ese caso al comer el primer fantasma se consiguen 200 puntos, 400 puntos con el segundo, 800 puntos con el tercero y 1600 puntos con el cuarto. En el estado *comestible* los fantasmas cambiarían a color azul.
   - El movimiento de los fantasmas puede ser más o menos complejo.
     - Generar aleatoriamente la dirección de entre las posibles direcciones que se pueden tomar en un momento determinado, pero teniendo en cuenta que no se permite cambiar de sentido (o sea si antes iba hacia la derecha no se permite que ahora vaya hacia la izquierda).
     - Ordenar las cuatros direcciones de movimiento según la distancia al comecocos. Luego las dos primeras direcciones se intercambiarían aleatoriamente con probabilidad 0.5. Finalmente se tomaría la primera dirección posible del anterior orden, respetando la restricción de que no se permite cambiar de sentido, al igual que antes.
   - El movimiento de los fantasmas cuando están en estado *comestible* podría ser en dirección contraría a como lo haría si estuviese en estado normal.
   - Cuando un fantasma alcanza al comecocos, éste muere.
   - Pueden asignarse tres vidas al comecocos, de forma que mientras no se agoten las tres, el juego no acabaría.

7. Como modificación a las opciones básicas, cuando el comecocos come todos los puntos de un nivel, se pasaría al siguiente nivel, en el que se podría incrementar un poco la velocidad de movimiento.

8. Incluir en el menú Juego las opciones de Salvar la partida, y la de Abrir una partida.

9. Incluir un menú de Ayuda.

10. Incluir un registro de los records.

11. Si se implementa la anterior opción, almacenar los records en un fichero, para que no se pierdan cuando se sale del programa.

12. Mostrar el tiempo que ha pasado desde que comenzó el juego. Para ello puede emplearse una nueva hebra con un bucle infinito, en el que periódicamente se mira la hora actual, y si ha cambiado respecto a la hora mostrada en el interfaz, se actualiza en él. Tras esta comprobación, la hebra debería ponerse a dormir durante un pequeño intervalo de tiempo.

13. Asociar un tiempo máximo para completar un nivel, que podría aparecer visible en alguna parte del interfaz. En el nivel 1, podemos poner por ejemplo 90 segundos. El tiempo reservado para completar cada nivel podría disminuirse a medida que pasamos de nivel.

14. El tiempo sobrante al superar un nivel puede usarse para sumar a los puntos conseguidos en ese nivel. Por ejemplo podríamos sumar la cantidad $puntosnivel = segundos * 10$.

4 Material a entregar

La entrega de la práctica debe constar de:

- **Análisis del problema**:

  El análisis del problema debe entregarse en un documento en formato pdf cuyo nombre debe ser `informe.pdf`. El documento irá identificado en una portada por medio del nombre del alumno. Contendrá un índice, al principio del documento, y, en su caso, una bibliografía al final.

  Este informe debe incluir:

  - **Diagrama de clases en UML**. Para hacer el diagrama de clases, puede emplearse una herramienta como **dia** (`http://projects.gnome.org/dia/`, o bien **umbrello** (http://uml.sourceforge.net/).
  - Justificación de la solución, convenientemente explicada, mediante la explicación de qué representa cada clase usada y detallando una por una todas las relaciones existentes entre las clases.
  - En caso de que se hayan implementado algunas de las mejoras, se explicarán con más detalle en una sección independiente del informe llamada *Mejoras implementadas*.

- **Manual de usuario y compilación del programa**.

- **Proyecto netbeans** completo dónde esté incluido todo el código fuente, aunque no es necesario que esté compilado (no entregar los ficheros .jar, y .class).

  El código debe estar documentado usando `javadoc`. Para ello es recomendable leer el documento sobre `javadoc` disponible en `http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html`. Se requiere al menos añadir comentarios para:

  - Indicar qué representa cada clase.
  - Qué representa cada dato miembro.
  - Qué hace cada método (o constructor), qué valor devuelve, y qué representa cada uno de sus parámetros.

- **Documentación** `html` generada por `javadoc`

Todos estos ficheros (informe con el análisis del problema, proyecto netbeans con el código fuente y subdirectorio de javadoc) serán empaquetados en un archivo formato `tar.gz` que se llamará exactamente `practica8.tar.gz` y se entregará mediante la plataforma docente antes del fin del plazo establecido.