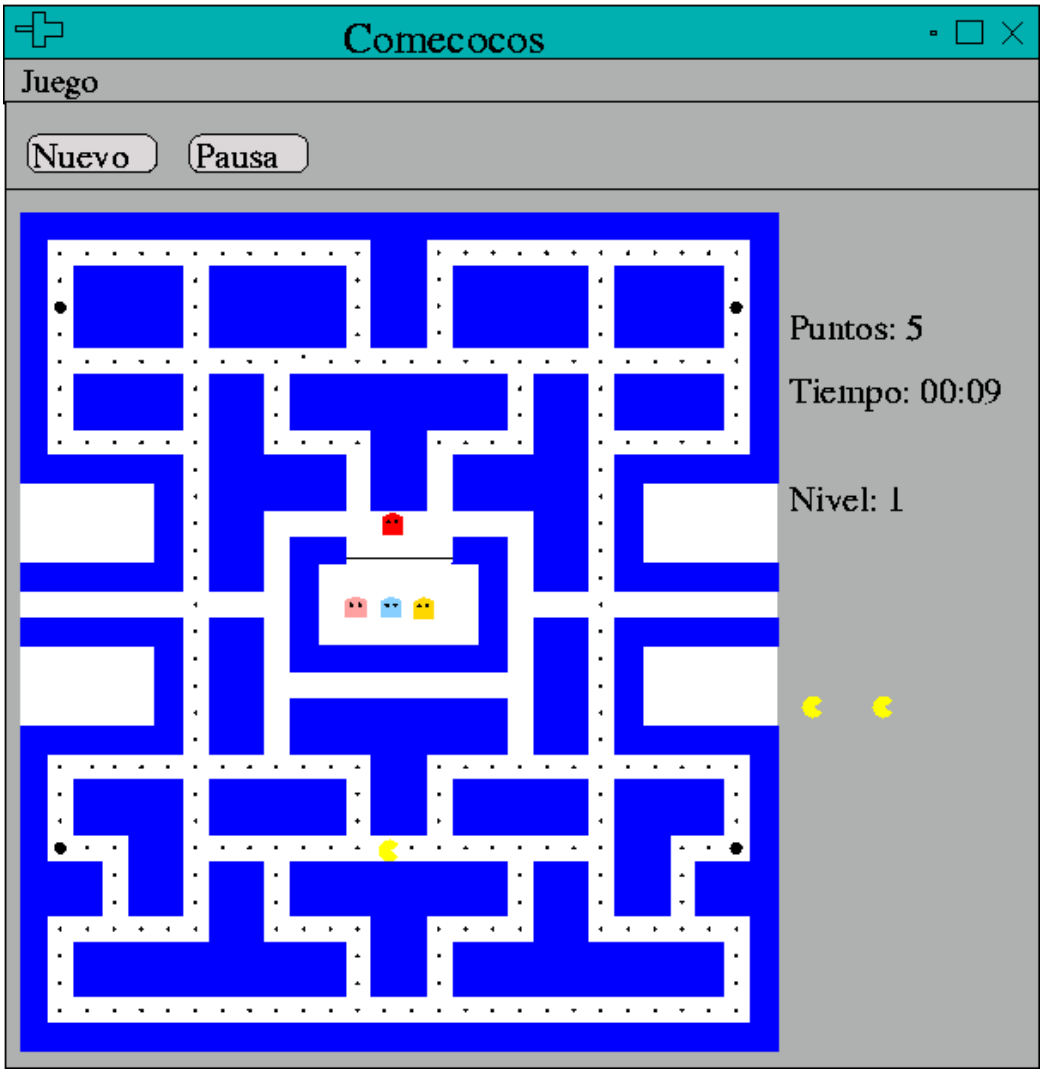
Nuevas tecnologías de la programación
Práctica 3: Juego del comecocos en Java (curso 2011-2012)

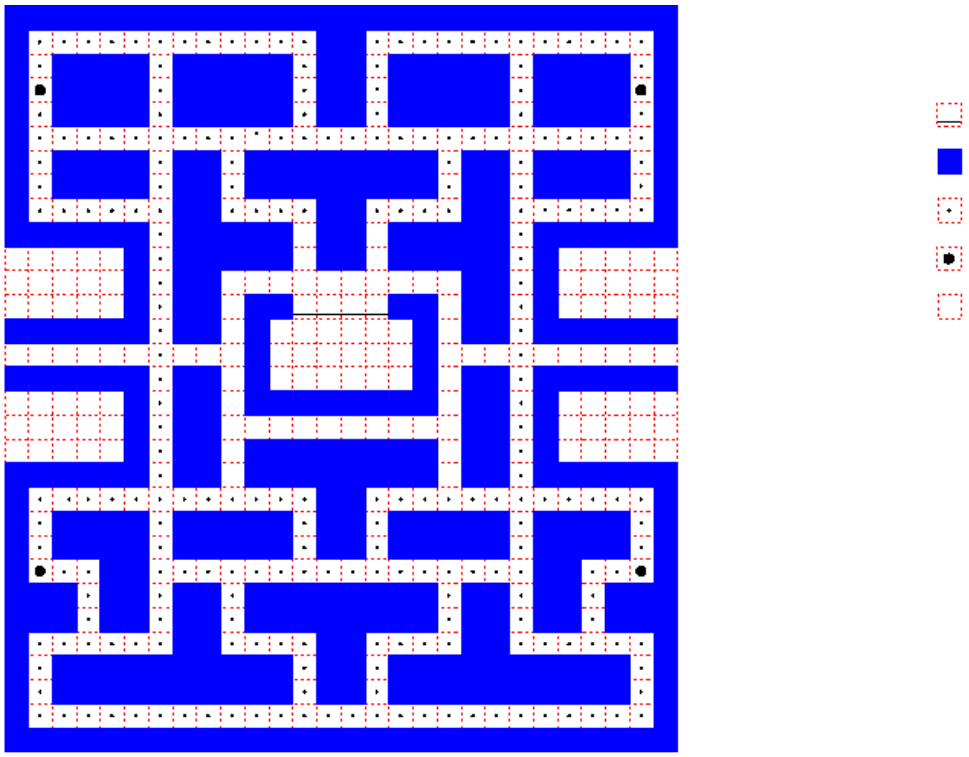
Descripción

La práctica consiste en la implementación en Java utilizando el kit de desarrollo j2sdk (se recomienda utilizar la versión j2sdk1.6) 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`. 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 en el programa, 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.



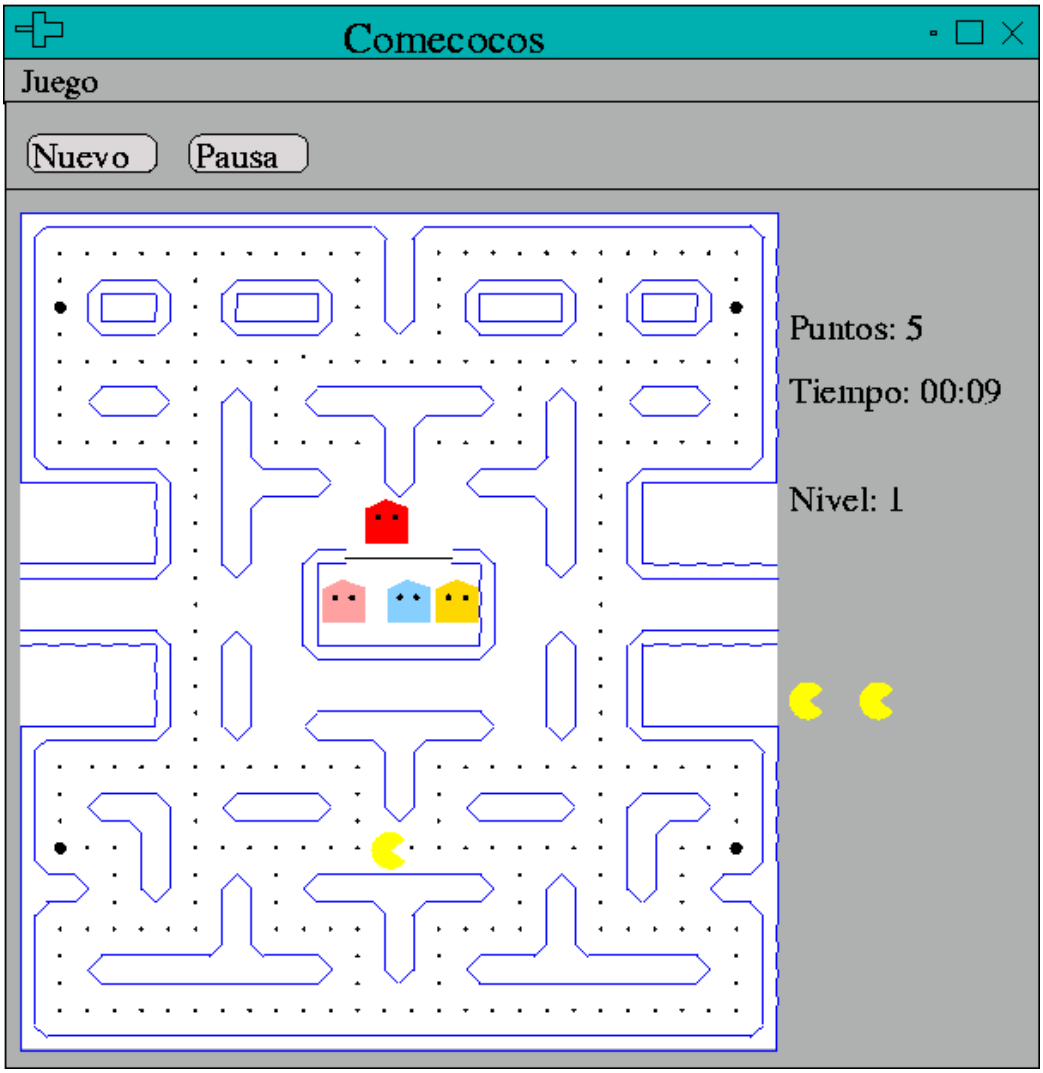
Para definir el estado inicial de la rejilla podriamos utilizar un vector de `String` de la siguiente forma:

```
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 vector codifica las seis primeras filas del laberinto, usando la siguiente codificación:

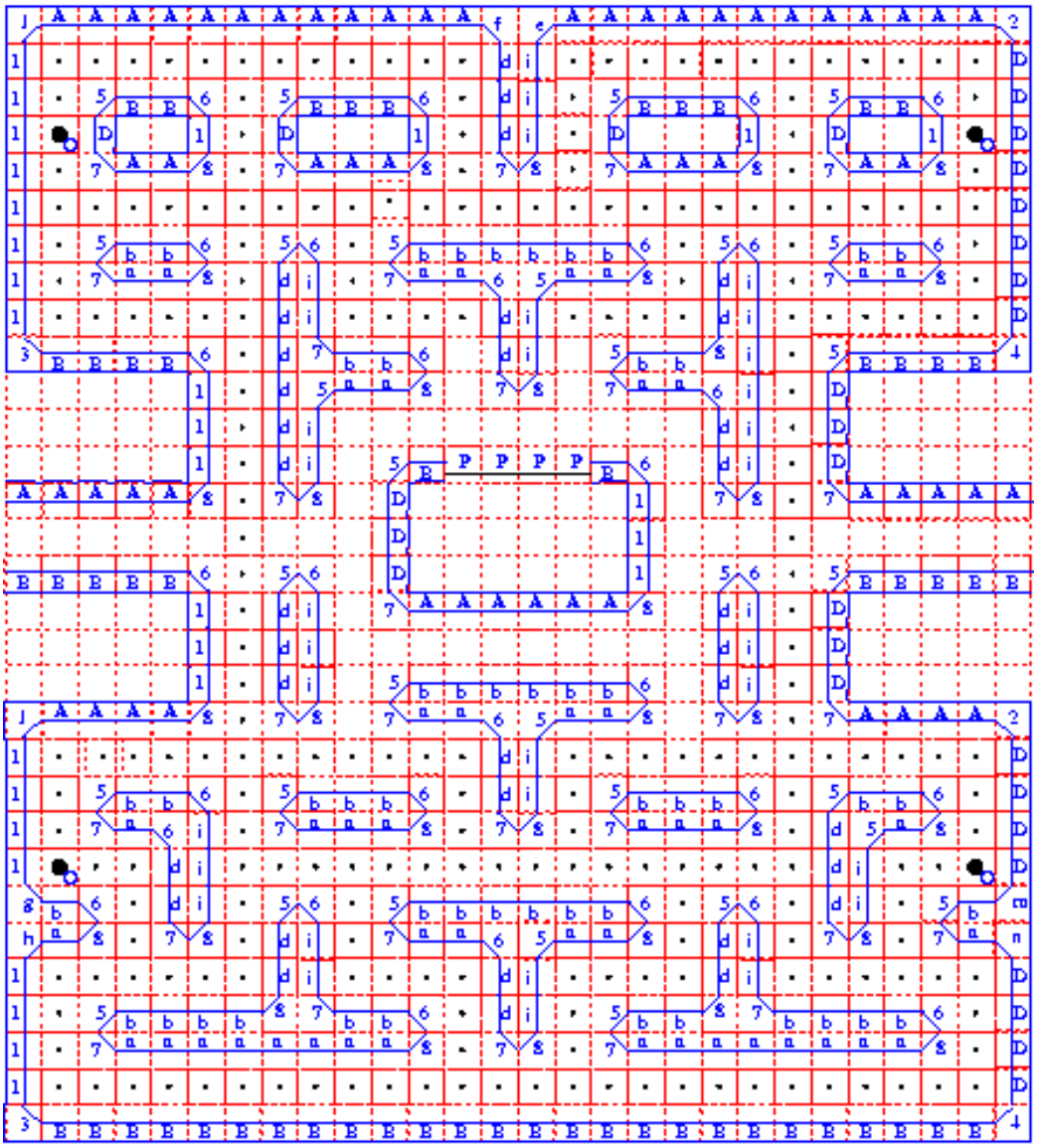
- Bloque: `B`
- Punto pequeño: `.`
- Punto grande: `o`

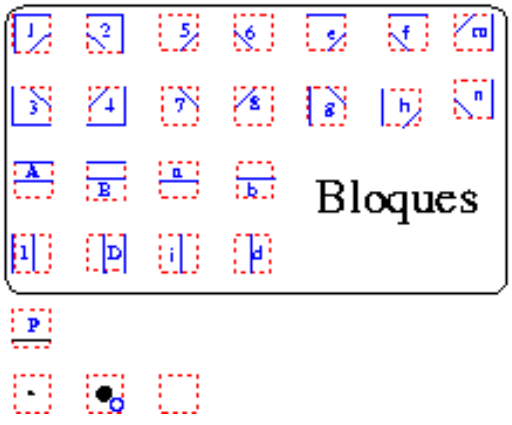
Podríamos mejorar el aspecto del laberinto si utilizamos distintos tipos de celdas para los bloques, según muestra la siguiente figura:



Mostramos de nuevo la rejilla, junto con los tipos de celdas necesarios 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"
};
```




Bloques

Requerimientos mínimos
Los requerimientos mínimos que se piden al programa son los siguientes:

1. Debe haber alguna forma de comenzar un nuevo juego y salir del programa (opciones de un menú o bien botones).
2. Sólo es obligatorio que aparezca el comecocos. Los fantasmas no son obligatorios.
3. No es obligatorio que aparezcan en el tablero los puntos pequeños y puntos grandes.
4. El comecocos puede dibujarse con un círculo relleno en color amarillo.
5. Como opción básica, el movimiento del comecocos será celda a celda.
6. Una vez que el comecocos comience a moverse, seguirá haciendolo en la dirección actual sin necesidad de pulsar ninguna tecla, mientras no encuentre un obstáculo.
7. Deben controlarse las teclas de cursor (izquierda, derecha, arriba y abajo) del teclado para cambiar la dirección de movimiento del comecocos.
8. En la opción básica el laberinto se dibujará en la primera forma (la más sencilla).

Opcionalmente se pueden incluir en el programa otras **mejoras**, explicándolas en la documentación del programa. 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 donde se encuentran encerrados inicialmente. La velocidad de los fantasmas debe ser algo menor a la del comecocos.

   - 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.
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 $puntosnivel = segundos * 10$.

Documentación a entregar

El programa y la documentación se entregará a través de la plataforma de docencia del departamento de Ciencias de la Computación e I.A. (`http://decsai.ugr.es`) enviando un fichero tar comprimido (practica3.tgz). Sólo es necesario que lo envíe uno de los autores de la práctica. En linux, la forma de crear el fichero tar comprimido es con el comando tar:

```
tar zcvf practica3.tgz CarpetaConTodo
```

El fichero debe contener el código fuente, makefile, ejecutable para linux y cualquier otro fichero que sea necesario para compilar el programa.
IMPORTANTE: El fichero debe contener también un fichero `Readme.txt` (fichero ASCII) cuyo contenido sea los apellidos y nombre de cada uno de los autores de la práctica, así como la dirección e-mail de cada uno de ellos (una línea pora cada autor).

El fichero también debe contener la documentación de la práctica. Tal documentación debe entregarse en los formatos postcript o bien pdf. Esta documentación contendrá:

1. Enunciado del problema.
2. Análisis del problema: requerimientos, modularidad, estructuras de datos, justificación de la solución, diseño de los algoritmos, etc.
3. Manuales de usuario y compilación del programa.

Tiempo de realización

El tiempo estimado para la realización de la práctica es de 8 horas. O sea de 4 semanas.

Fecha de entrega

La práctica puede ser enviada por correo hasta el día 17 de febrero de 2012. Al día siguiente se publicará en la página web de la asignatura (`http://decsai.ugr.es/~acu/NTP`) la lista de alumnos que han enviado la práctica.