Práctica 8: El juego del comecocos en Java

CP 3º GITT

David Ramírez Sierra

ÍNDICE

1. Introducción

1.1 Descripción/Enunciado:

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.

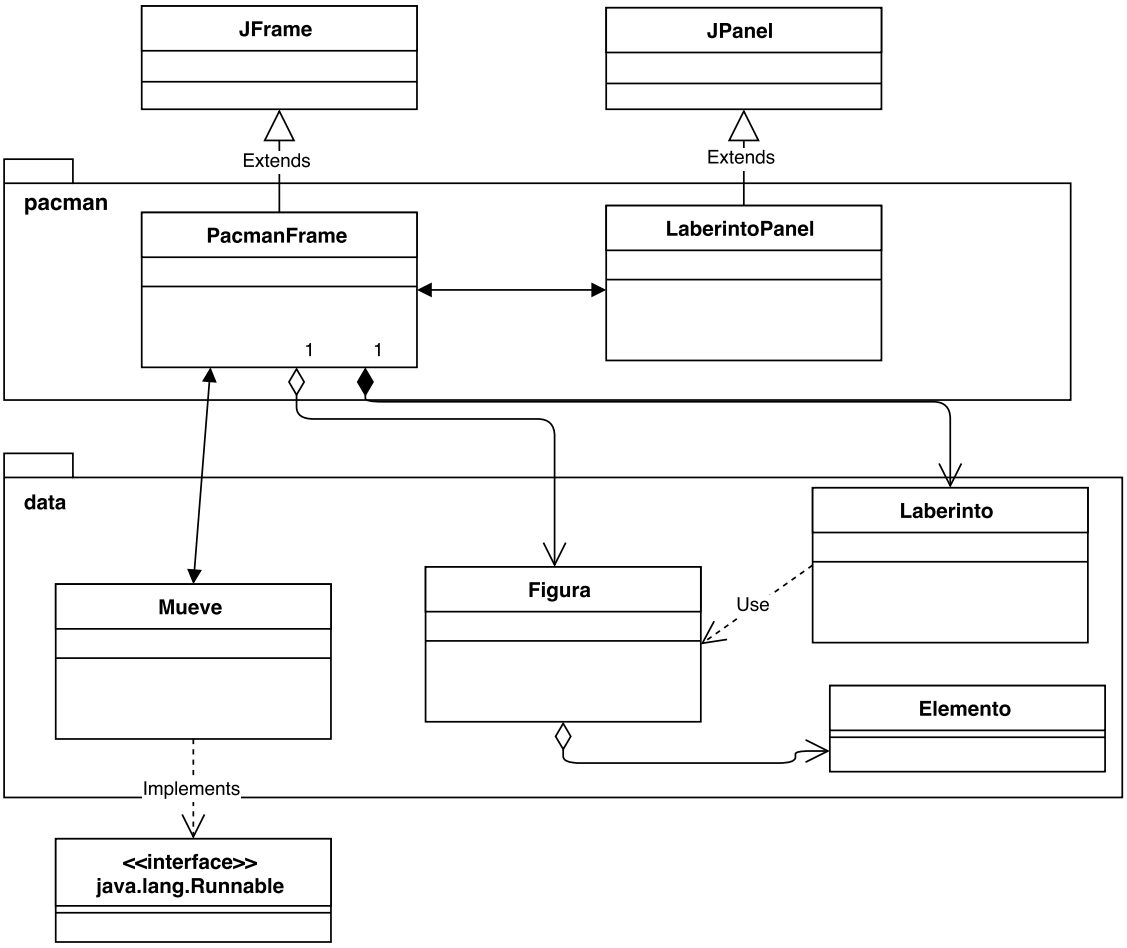
1.2 Requerimientos mínimos:

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

- El mapa del laberinto debe definirse a partir de un array de Strings según se ha indicado anteriormente. ✓
- Debe haber alguna forma de comenzar un nuevo juego y salir del programa (opciones de un menú o bien botones). ✓
- Sólo es obligatorio que aparezca el comecocos. Los fantasmas no son obligatorios. ✓
- No es obligatorio que aparezcan en el tablero los puntos pequeños y puntos grandes. ✓
- El comecocos puede dibujarse con un círculo relleno en color amarillo. ✓
- Como opción básica, el movimiento del comecocos será celda a celda. ✓
- 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. ✓
- 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. ✓
- Deben controlarse las teclas de cursor (izquierda, derecha, arriba y abajo) del teclado para cambiar la dirección de movimiento del comecocos. ✓
- En la opción básica, el laberinto se dibujará en la primera forma (la más sencilla). ✓

2. Análisis del Problema

2.1 Diagrama de Clases UML.



A continuación, se detallan tanto los atributos como los métodos de cada clase:

LaberintoPanel
- frame : PacmanFrame - anchoCelda: int
+ LaberintoPanel() +LaberintoPanel(PacmanFrame fr) + dibujaRejilla(java.awt.Graphics g) # paintComponent(Graphics g) + dibujaFigura(Figura fig, java.awt.Graphics g) - keyPressed(java.awt.event.KeyEvent evt) - mouseEntered(java.awt.event.MouseEvent evt) - keyPressed2(java.awt.event.KeyEvent evt)

PacmanFrame
- rejilla : Laberinto

- figura: Figura - mueve : Mueve - ini : boolean - cronometroActivo : boolean
+PacmanFrame() -parar_botonActionPerformed(java.awt.event.ActionEvent evt) -salir_botonActionPerformed(java.awt.event.ActionEvent evt) -salir(java.awt.event.KeyEvent evt) -jMenu1MouseClicked(java.awt.event.MouseEvent evt) -jMenu2MouseClicked(java.awt.event.MouseEvent evt) + setPuntuacion(int score) + getRejilla() : Laberinto +getFigura() : Figura + nuevaFigura() + inicializaJuego() + getPanel(): LaberintoPanel + iniciarCronometro() + pararCronometro()

Elemento
- fila : int - columna: int
+ Elemento(int f,int c) + getFila(): int + getColumna(): int + setFila(int valor) + setColumna(int valor)

Figura
- elements : Vector<Elemento> - xorigen: int - yorigen: int - tipo: int
+ Figura(int fila0) + nuevaFigura() : Figura + getTipo() : int + setDireccion(int dir) - addElements(int fila, int valor) +getNElements(): int +getElementAt(int pos): Elemento + getXOrigen(): int + getYOrigen(): int + mueve(int direccion)

Laberinto
- anchura : int - altura: int - celdas: int[][]
+ Laberinto(int w,int h) + getAnchura(): int + getAltura(): int + assignTipoCelda(int x,int y,int valor) + getTipoCelda(int x,int y): int

+ setTipoCelda(int x,int y,int tipo) + initRejilla() + copiaFiguraEnRejilla(Figura fig): boolean + seChoca(Figura fig, int direccion): boolean

Mueve
- delay : int - continuar : boolean - suspendFlag: boolean - frame: PacmanFrame - score: int
+ Mueve(PacmanFrame fr,int nivel) + inicializaScore() + run() + suspender() + reanudar() + parar() + getParado() : boolean + actualizaRetardo(int nivel): int

2.2 Justificación de la Solución

El juego está comprendido en dos paquetes: data y pacman. En data se encuentran todas las clases relacionadas con el laberinto, la figura y el movimiento del juego. En pacman están las clases que crean y hacen posible el transcurso del juego, así como su dibujado y la interfaz gráfica. Por tanto, es dentro de este paquete donde se encuentra la clase principal.

Una de las clases del paquete data, es Elemento. Esta clase representa el elemento mínimo del cual está formada una Figura. Por tanto, contiene como datos miembro la columna y la fila donde este se encuentra. Por otro lado, la clase Figura, tiene como principal atributo un vector de Elementos que son los encargados de componer dicha Figura. En Figura también podemos encontrar la dirección que sigue la figura y las constantes finales que representan las diferentes direcciones. Es decir, todo lo referente a la composición de la figura del comecocos se da en la clase Figura.

Por otro lado, también dentro de el paquete data, tenemos la clase Laberinto. En esta clase se realizan dos funciones principalmente: inicializar el laberinto del juego a partir de un array de String, asignando a cada bloque su correspondiente tipo (vacía, muro, punto...) y comprobar si el comecocos se choca con alguna de las paredes a partir de conocer la figura y su dirección. Por tanto, es necesario instanciar la clase Figura para este método, de manera que existe una relación de dependencia o instanciación tal y como se ha representado en el diagrama UML.

Por último, dentro del paquete data, también podemos encontrar la clase Mueve. Esta clase representa una Tarea, por tanto, implementa el interfaz Runnable. En dicha tarea se realiza el movimiento continuo del comecocos, siempre y cuando la figura no se choque. Además, también es dentro de este método sobreescrito “run()” donde se

comprueba el tipo de celda, para saber si se ha comido un punto, etc. Asímismo, esta clase cuenta con los métodos sincronizados necesarios para controlar dicha hebra y el bucle que contiene el run() para parar, reanudad o suspender el movimiento haciendo uso de banderas booleanas.

En el otro paquete llamado pacman, encontramos las clases PacmanFrame y LaberintoPanel. LaberintoPanel extiende de JPanel ya que es comprende un elemento gráfico. Además, cuenta con un objeto de tipo PacmanFrame como uno de sus miembros privados. Esta relación es recíproca por lo que existe una estrecha relación de asociación uno a uno entre ambas clases. Una de las funciones de esta clase es ir redibujando cada componente según corresponda en concordancia con los cambios que se dan el juego. Por tanto sus métodos son dibujar la rejilla y dibujar la figura. Otra de las funciones de esta clase es recoger el movimiento de las flechas del teclado para cambiar la dirección de la figura. Es importante destacar, que en este método la figura no realiza movimiento, solamente cambia su dirección.

Por último, la clase principal PacmanFrame es la responsable de crear el laberinto, crear la figura, iniciar el movimiento y empezar la partida, además de ser la principal interfaz gráfica. Por tanto, tres de sus datos miembros son instancias de las clases Laberinto, Mueve y Figura. Se ve con claridad la relación tan fuerte que existe entre dichas clases. En esta clase también se maneja a través de botones la parada del juego o el comienzo de una nueva partida.

2.3 Mejoras

Las mejoras que han sido implementadas son las siguientes:

Mejora 1:

Se pueden incluir los puntos pequeños y grandes en el tablero de forma que:

- *Cuando el come cocos pasa por encima de un punto pequeño grande este 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.*

En primer lugar, se han definido dos nuevos tipos de celda en la clase laberinto que representan el punto pequeño y el punto grande, de manera que en la inicialización de Laberinto se han incluido estos nuevos elementos y la implementación de dibujaRejilla(java.awt.Graphics g) de la clase LaberintoPanel también lo tendrá en cuenta:

```
public static final int PUNTO         = 3;
public static final int PUNTO_GRANDE    = 4;
```

Además, en la tarea que se representa en la clase Mueve, donde se realiza el movimiento, se hace un análisis de la celda y si es de tipo PUNTO o PUNTO_GRANDE , el comecocos se "come" el punto, es decir, se cambia ese tipo de celda por una celda VACÍA. Por otro lado, también se incrementa como corresponde el atributo privado entero llamado score que es el que lleva el recuento de los puntos.

Mejora 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.

Para ello, se han incluido dos botones en la clase principal PacmanFrame, uno para detener y reanudar el juego, y otro para comenzar una nueva partida.
De lo que se trata es de llamar a los diferentes métodos de control de la hebra del objeto mueve del tipo Mueve. De manera que se puede reanudar o suspender

Mejora 5 :

Se pueden utilizar diferentes aspectos para el comecocos según sea su dirección de movimiento.

Para la implementación de este apartado, he modificado el método dibujarFigura de la clase LaberintoPanel. Para darle diferentes aspectos al comecocos, se dibuja y rellena un arco con la función drawArc de java. Dependiendo de la dirección que sigue la figura, así será la orientación que siga ese arco. Para ello se multiplica por un factor ("b") que depende de la dirección. A continuación, se muestra una imagen aclarativa:

```
int b = 0;
switch (frame.getFigura().direction) {

    case Figura.ABAJO:
        b = 12;
        break;
    case Figura.ARRIBA:
        b = 3;
        break;
    case Figura.DERECHA:
        b = 17;
        break;
    case Figura.IZQUIERDA:
        b = 8;
        break;


}
g.fillArc(xoffset + elemento.getColumna() * anchoCelda, yoffset + elemento.getFila() * anchoCelda,
        anchoCelda, anchoCelda, anchoCelda * b, anchoCelda * 4);

g.setColor(Color.RED);
g.drawArc(xoffset + elemento.getColumna() * anchoCelda, yoffset + elemento.getFila() * anchoCelda,
        anchoCelda, anchoCelda, anchoCelda * b, anchoCelda * 4);
```

Se puede ver que para cada dirección, corresponde un factor diferente.

Mejora 9 :

Incluir un menú de Ayuda.

Mejora 12:

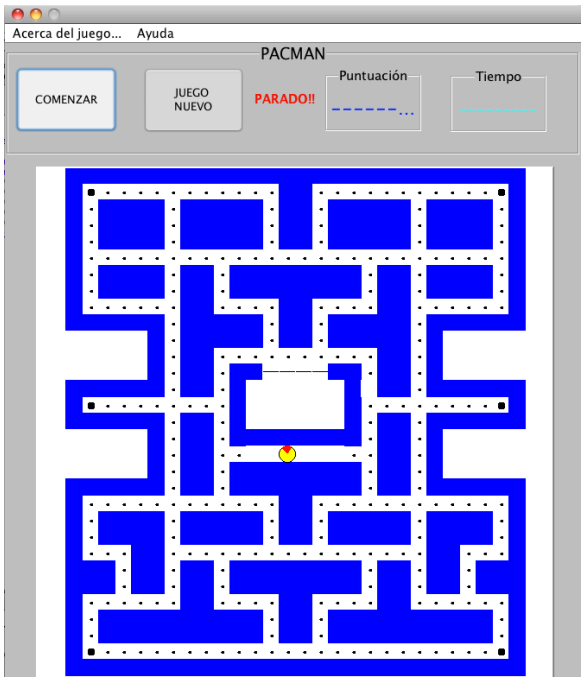
Mostrar el tiempo que ha pasado desde que comenzó el juego.

Para mostrar el tiempo, he creado una nueva tarea en la clase PacmanFrame y en la que dentro del método sobreescrito run() se va contabilizando el tiempo. Dentro de un bucle while se va durmiendo la hebra cada 100 milisegundos, de manera que se va contabilizando de 100 en 100 milisegundos. El algoritmo es muy sencillo. Para ello he creado varios métodos en los que se puede iniciar o parar el cronómetro. Asímismo, para el inicio de una partida las variables correspondientes al tiempo se inicializan de nuevo a 0.

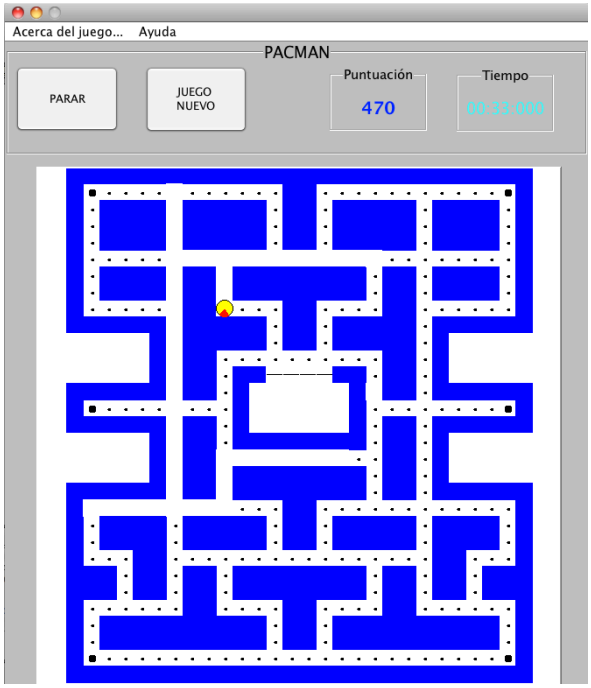
```
public void run(){
    //minutos = 0 ; segundos = 0; milesimas = 0;
    //min es minutos, seg es segundos y mil es milesimas de segundo
    String min="", seg="", mil="";
    try
    {
        //Mientras cronometroActivo sea verdadero entonces seguira
        //aumentando el tiempo
        while( cronometroActivo )
        {
            Thread.sleep( 100 );
            //Incrementamos 4 milesimas de segundo
            milesimas += 100;

            //Cuando llega a 1000 osea 1 segundo aumenta 1 segundo
            //y las milesimas de segundo de nuevo a 0
            if( milesimas == 1000 )
            {
                milesimas = 0;
                segundos += 1;
                //Si los segundos llegan a 60 entonces aumenta 1 los minutos
                //y los segundos vuelven a 0
                if( segundos == 60 )
                {
                    segundos = 0;
                    minutos++;
                }
            }

            //Esto solamente es estetica para que siempre este en formato
            //00:00:000
            if( minutos < 10 ) min = "0" + minutos;
            else min = minutos.toString();
            if( segundos < 10 ) seg = "0" + segundos;
            else seg = segundos.toString();

            if( milesimas < 10 ) mil = "00" + milesimas;
            else if( milesimas < 100 ) mil = "0" + milesimas;
            else mil = milesimas.toString();

            //Colocamos en la etiqueta la informacion
            jLabel3.setText( min + ":" + seg + ":" + mil );
        }

    }catch(Exception e){}
}
```

3. Manual de Usuario

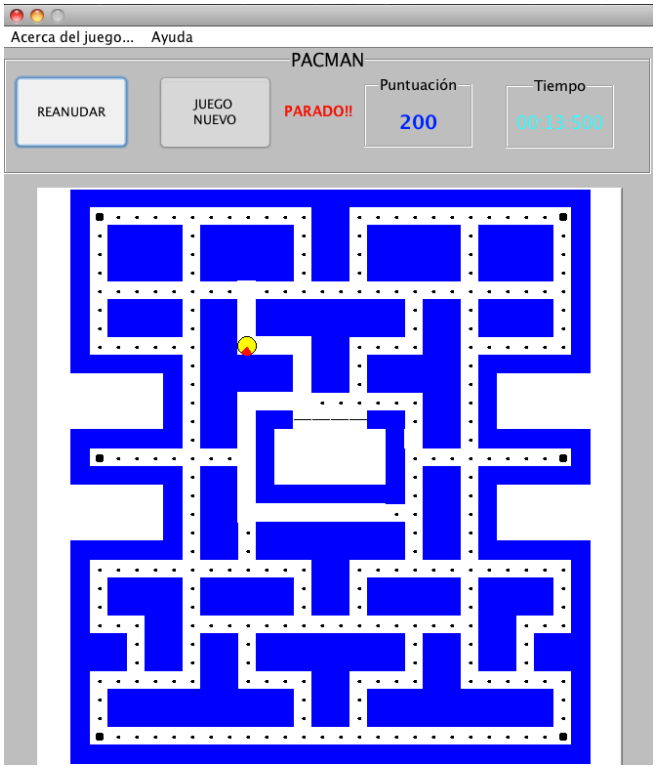
Una vez compiladas todas las clases en sus respectivos paquetes, el juego se ejecuta en la clase principal PacmanFrame. Una vez ejecutado aparece la siguiente ventana:



Como vemos, al abrir el juego, este se encuentra en parado, encontrándose el comecocos situado en posición inicial.
Para dar comienzo al juego es necesario pinchar en el botón comenzar, de manera que ya será posible controlar el comecocos y tanto el tiempo como la puntuación irán variando como corresponda.



En mitad de la partida, es posible parar el juego pulsando el botón de parar, con lo que el tiempo dejará de contar y será imposible mover el comecocos. De igual forma, si el juego se encuentra en estado "parado", será posible reanudarlo pulsando el botón de reanudar.



Otra de las posibles acciones que se pueden realizar es crear un Juego Nuevo, pulsando en el botón de Juego Nuevo. De manera que el juego inicializará la puntuación y el contador de tiempo a 0. Y tanto el comecocos como los cocos se inicializarán según el laberinto inicial.

Además de esto, hay un menú donde se pueden encontrar ventanas emergentes con información Acerca del Juego y Ayuda.