Práctica 1: Simulación de un Mundo Celular Simple

Fecha de entrega: 12 de Noviembre de 2015 a las 23:00

Objetivo: Iniciación a la orientación a objetos y a Java; uso de arrays; manipulación de cadenas con la clase `String`; entrada y salida por consola.

1. Introducción

En esta práctica vamos a realizar una simulación de un mundo celular muy simple que evoluciona a lo largo de una superficie. La aproximación más conocida a este tipo de simulaciones es el *juego de la vida* diseñado por el matemático británico John Horton Conway en 1970. El juego de la vida es un juego de cero jugadores, donde se muestra la evolución de un mundo celular inicial, que evoluciona siguiendo unas determinadas reglas. En este juego las células no se mueven y son estáticas. Lo único que puede ocurrir es que las células nazcan o se mueran dependiendo del número de células vecinas que tengan. Para más información sobre el juego de la vida se puede consultar, por ejemplo, `https://es.wikipedia.org/wiki/Juego_de_la_vida#Ejemplos_de_patrones`.

En esta primera práctica vamos a implementar una variación del juego de la vida donde las células se mueven, además de morir o nacer, de acuerdo a unas normas concretas. La simulación de nuestro mundo celular la haremos a través de comandos, tal y como describimos a continuación.

2. Descripción de la práctica

Nuestro mundo vamos a representarlo como una matriz de células (superficie). Las células se mueven de forma aleatoria a alguna posición libre de sus posiciones vecinas. Las posiciones vecinas de una célula intermedia son ocho, pero lógicamente las células colocadas en los laterales de la superficie tienen menos posiciones vecinas y hay que tratarlas con más cuidado. Por ejemplo, la célula `X` en la figura de la izquierda tiene como posiciones vecinas

las marcadas con L, es decir ocho posiciones. Sin embargo en la figura central y derecha el número de posiciones vecinas decrece.

```
_ L L L _     _ _ _ L X     L L _ _ _
_ L X L _     _ _ _ L L     X L _ _ _
_ L L L _     _ _ _ _ _     L L _ _ _
```

Las reglas de movimiento, nacimiento, defunción y reproducción de una célula son las siguientes:

- Una célula se mueve a una posición vecina aleatoria siempre que pueda, es decir que haya alguna posición libre.
- Si una célula está MAX_PASOS_SIN_MOVER pasos sin moverse, entonces la célula muere por falta de ejercicio.
- Cuando una célula ha dado PASOS_REPRODUCCION pasos, entonces pueden pasar dos cosas:
  - Si la célula tiene alguna posición vecina libre, entonces esta célula se mueve aleatoriamente a una de ellas, dejando una cría en su posición, es decir una nueva célula.
  - Si la célula no puede moverse entonces muere.

Podemos visualizar y controlar la evolución del mundo mediante instrucciones en lenguaje natural. En esta primera versión tendremos las siguientes instrucciones:

- PASO: Realiza un movimiento sobre cada una de las células del mundo, respetando las reglas de evolución de éste.
- INICIAR: Inicia la simulación. Es decir inicializa el mundo eliminando las células anteriores de la superficie e introduciendo nuevas células en posiciones aleatorias. El número de células vendrá especificado por una constante.
- CREARCELULA f c: Crea una nueva célula y la introduce en el mundo en la posición (f,c). Si la posición está ocupada manda un mensaje de error.
- ELIMINARCELULA f c: Elimina la célula del mundo en la posición (f,c). En caso de que dicha posición estuviera vacía manda un mensaje de error.
- AYUDA: Muestra los distintos comandos que se pueden ejecutar en la Práctica 1, junto con una breve descripción de lo que hacen.
- VACIAR: Elimina todas las células del mundo.
- SALIR: Es una metainstrucción que nos permite abandonar la simulación

Si introducimos una instrucción distina a las anteriores el programa te lo indicará con un mensaje.

3. Ejemplo de ejecución

A continuación mostramos un ejemplo de ejecución de nuestra simulación. La notación `N-M` indica que en esa posición hay una celula a la que la quedan `M` pasos para reproducirse y `N` pasos en los que puede estar sin moverse. La configuración inicial tiene `3` células. Observa que las instrucciones *no* son sensibles a mayúsculas ni minúsculas (puedes escribirlas indistintamente).

```
 -    -   1-3
 -    -    -
 -   1-3  1-3

Comando > PaSo
Movimiento de (0,2) a (1,2)
Movimiento de (2,1) a (1,1)
Movimiento de (2,2) a (2,1)

  -    -    -
  -   1-2  1-2
  -   1-2   -

Comando > paso
Movimiento de (1,1) a (0,1)
Movimiento de (1,2) a (1,1)
Movimiento de (2,1) a (2,0)

  -   1-1   -
  -   1-1   -
 1-1   -    -

Comando > paso
Movimiento de (0,1) a (0,0)
Nace nueva celula en (0,1) cuyo padre ha sido (0,0)
Movimiento de (1,1) a (0,2)
Nace nueva celula en (1,1) cuyo padre ha sido (0,2)
Movimiento de (2,0) a (1,0)
Nace nueva celula en (2,0) cuyo padre ha sido (1,0)

 1-3  1-3  1-3
 1-3  1-3   -
 1-3   -    -

Comando > paso
Movimiento de (0,1) a (1,2)
Movimiento de (0,2) a (0,1)
Movimiento de (1,0) a (2,1)
Movimiento de (1,1) a (1,0)
Movimiento de (2,0) a (1,1)

 0-2  1-2   -
 1-2  1-2  1-2
  -   1-2   -

Comando > paso
Muere la celula de la casilla (0,0) por inactividad
```

```
Movimiento de (0,1) a (0,0)
Movimiento de (1,0) a (0,1)
Movimiento de (1,1) a (2,2)
Movimiento de (1,2) a (0,2)
Movimiento de (2,1) a (1,0)

 1-1  1-1  1-1
 1-1   -    -
  -    -   1-1

Comando > Comando > crearCelula 0 2
Imposible crear una nueva celula.. Posicion ocupada

 1-1  1-1  1-1
 1-1   -    -
  -    -   1-1

Comando > crearCelula 2 0
Creamos nueva celula en la posición: (2,0)

 1-1  1-1  1-1
 1-1   -    -
 1-3   -   1-1

Comando > iniciar
Iniciando simulacion.....

  -    -    -
  -    -   1-3
 1-3  1-3   -

Comando > ayuda
POSIBLES COMANDOS:
  PASO:  realiza un paso en la simulacion
  AYUDA: muestra esta ayuda
  SALIR: cierra la aplicación
  INICIAR: inicia una nueva simulación
  VACIAR: crea un mundo vacío
  CREARCELULA F C: crea una nueva celula en la posición (f,c) si es posible
  ELIMINARCELULA F C: elimina una celula de la posición (f,c) si es posible

  -    -    -
  -    -   1-3
 1-3  1-3   -

Comando > vaciar
Vaciando la superficie....
  -    -    -
  -    -    -
  -    -    -

Comando > salir
Fin de la simulacion.....
```

4. Clases que hay que implementar

Para crear esta primera aplicación en Java vamos a implementar las siguientes clases:

- Celula: Representa una célula del mundo. Contiene atributos privados para contabilizar el *número de pasos en los que la célula no se ha movido* y el *número de pasos dados* (tanto si se ha movido como si no) realizados en el mundo.

- Superficie: Es la superficie donde transcurre la evolución de las células. La superficie la vamos a representar mediante una matriz de células, por lo tanto esta clase contendrá un atributo `private Celula[][] superficie`, junto con los atributos `private int filas` y `private int columnas` que determinan el tamaño de la superficie.

  La *constructora* de esta clase `public Superficie(int nf, int nc)` inicializa los atributos de la misma, colocando en todas las posiciones de la matriz `superficie` el valor `null` (que indica que no hay ninguna célula en la posición).

- Mundo: El mundo contiene a la superficie y por lo tanto tiene el atributo `private Superficie superficie`. La constructora por defecto de esta clase inicializa la `superficie` colocando un número de células (dado por una constante) en posiciones aleatorias de la `superficie`.

  Esta clase tiene el método `public void evoluciona()`, que da un paso completo en la evolución, es decir para cada célula de la superficie ejecuta un paso de acuerdo a las reglas que se describieron en la Sección 2. El método `evoluciona` recorre la `superficie` y dependiendo de las reglas de la vida, va pidiéndola a ésta que realice los pasos pertinentes. Ten en cuenta que este método es el que determina cómo se mueven las células, las coloca, comprueba si mueren o se reproducen, etc.. Para ello tendrá que dar las órdenes pertienentes a la superficie para poder consultar la información sobre las células. Por otro lado cuando el mundo evoluciona, hay que controlar que una célula no se mueva dos veces en el mismo paso de evolución.

  Además del método anterior, harán falta métodos públicos para crear una célula en la posición `(f,c)` de la `superficie` o para eliminar la célula de la posición `(f,c)` de la `superficie`. Ambos métodos serán booleanos ya que no siempre será posible realizar la operación.

- Controlador: Esta clase contiene el intérprete de los posibles comandos que se pueden ejecutar en consola. Su método principal es `public void realizaSimulacion()`, que consiste en un bucle en el que se pide un comando al usuario y se ejecuta dicho comando. Además si el comando es incorrecto se muestra un mensaje de error. El bucle cuando el usuario teclea el comando `SALIR`.

  El controlador tiene como atributos privados `Mundo mundo` y `Scanner in`. El primero representa el mundo sobre el que ejecutar los comandos, y el segundo es un scanner para realizar las operaciones de lectura. La constructora del `Controlador` tiene la forma `public Controlador(Mundo mundo, Scanner in)` y simplemente inicializa los atributos anteriores.

- Main: Es la clase que contiene el método `main` de la práctica. Es la responsable de crear el `Mundo` a través de su constructora por defecto, de crear el controlador invocando a su constructora con argumentos, y de realizar la simulación invocando al método `realizaSimulacion()` de la clase `Controlador`.

5. Entrega de la práctica

La práctica debe entregarse utilizando el mecanismo de entregas del campus virtual, no más tarde de la fecha y hora indicada en la cabecera de la práctica.

El fichero debe tener al menos el siguiente contenido:

- Directorio src con el código de todas las clases de la práctica.
- Fichero alumnos.txt donde se indicará el nombre de los componentes del grupo.
- Directorio doc con la documentación generada automáticamente sobre la práctica.