Práctica 2: Nuevas Células en Nuestro Mundo

Fecha de entrega: 11 de Diciembre de 2015 a las 23:00

Objetivo: Herencia: uso e implementación.

1. Nuevo tipo de células

En la Práctica 2 vamos introducir, además de las células de la Práctica 1, un nuevo tipo de células más sofisticadas. Estas nuevas células, que a partir de ahora llamaremos células complejas, se caracterizan porque se mueven de forma aleatoria por el mundo, a posiciones que pueden estar libres u ocupadas por otras células. Cuando una célula compleja se mueve a una posición libre no pasa nada, tan sólo se realiza el movimiento. Sin embargo si la posición a la que se mueve está ocupada por una célula de las de la Práctica 1 (a partir de ahora células simples) entonces se la come, es decir desaparece la célula simple. Finalmente las células complejas son muy amigas y no se atacan entre sí. Por lo tanto si la posición aleatoria elegida para mover una célula compleja está ocupada por otra célula compleja, entonces no se realiza el movimiento. Las células complejas tienen además una peculiaridad, y es que cuando comen más de MAX_COMER células simples, entonces explotan y se mueren. En este caso MAX_COMER representa una constante.

Al aparecer una nueva especie, algunos de los comandos de la Práctica 1 se modifican. Concretamente desaparece el comando CREARCELULA f c, y aparecen los nuevos comandos CREARCELULASIMPLE f c y CREARCELULACOMPLEJA f c, que se comportan como CREARCELULA, pero crean una célula del tipo especificado.

Para representar las células vamos a utilizar una jerarquía de clases, donde tendremos una clase abstracta Celula, de la que heredarán las clase concretas CelulaSimple y CelulaCompleja. Por otro lado vamos a reorganizar los comandos para construir una jerarquía de clases de forma que cada comando tenga entidad propia. En la Sección 3 especificaremos con detalle las clases necesarias para implementar la práctica.

2. Ejemplos de ejecución

En el siguiente ejemplo de ejecución partimos de un mundo en el que hay tres células simples y dos células complejas.

```
 -  -  -  X
 -  -  -  -
 -  -  *  -
 -  X  -  X
 -  -  -  *

Comando > paso
Movimiento de (0,3) a (0,2)
Celula Compleja en (2,2) se mueve a (0,2) --COME--
Movimiento de (3,1) a (2,1)
Movimiento de (3,3) a (2,2)
Celula Compleja en (4,3) se mueve a (0,1) --NO COME--

 -  *  *  -
 -  -  -  -
 -  X  X  -
 -  -  -  -
 -  -  -  -

Comando > paso
Celula Compleja en (0,1) se mueve a (3,3) --NO COME--
Celula Compleja en (0,2) se mueve a (1,0) --NO COME--
Movimiento de (2,1) a (3,2)
Movimiento de (2,2) a (1,2)
 -  -  -  -
 *  -  X  -
 -  -  -  -
 -  -  X  *
 -  -  -  -

Comando > paso
Celula Compleja en (1,0) se mueve a (2,1) --NO COME--
Movimiento de (1,2) a (1,1)
Nace nueva celula en (1,2) cuyo padre ha sido (1,1)
Movimiento de (3,2) a (2,2)
Nace nueva celula en (3,2) cuyo padre ha sido (2,2)
Celula Compleja en (3,3) se mueve a (1,2) --COME--

 -  -  -  -
 -  X  *  -
 -  *  X  -
 -  -  X  -
 -  -  -  -

Comando > paso
```

```
Movimiento de (1,1) a (0,1)
Celula Compleja en (1,2) se mueve a (4,1) --NO COME--
Celula Compleja en (2,1) se mueve a (3,2) --COME--
Movimiento de (2,2) a (1,2)

 -  X  -  -
 -  -  X  -
 -  -  -  -
 -  -  *  -
 -  *  -  -

Comando > paso
Movimiento de (0,1) a (0,0)
Movimiento de (1,2) a (2,1)
Celula Compleja en (3,2) se mueve a (0,0) --COME--
Explota la celula compleja en (0,0)
Celula Compleja en (4,1) se mueve a (0,3) --NO COME--

 -  -  -  *
 -  -  -  -
 -  X  -  -
 -  -  -  -
 -  -  -  -

Comando > crearcelulacompleja 0 0
Creamos nueva celula en la posición: (0,0)
 *  -  -  *
 -  -  -  -
 -  X  -  -
 -  -  -  -
 -  -  -  -

Comando > eliminarCelula 0 3
Eliminamos la celula de la posición: (0,3)

 *  -  -  -
 -  -  -  -
 -  X  -  -
 -  -  -  -
 -  -  -  -

Comando >
```

3. Clases que hay que implementar

Para implementar la Práctica 2 necesitaremos al menos las siguientes clases:

- Main: Clase similar a la de la Práctica 1.

- Comando: Es una clase abstracta pura, es decir que todos su métodos son abstractos. Concretamente contendrá los siguientes tres métodos:

```
public abstract void ejecuta(Mundo mundo);
public abstract Comando parsea(String[ ] cadenaComando);
public abstract String textoAyuda();
```

  El primer método ejecuta el comando correspondiente sobre el mundo. El segundo método recibe un array de String, que debe procesar devolviendo el comando que representa el string. Finalmente el tercer método devuelve un String con la información de ayuda que se quiera mostrar sobre el comando. De esta clase abstracta heredan tantas clases concretas como comandos hay en la Práctica 2. Concretamente tendremos nueve clases heredando de la clase Comando.

- ParserComandos: Esta clase es la encargada de parsear un array de String y construir el comando al que hace referencia dicho String. Esta clase contiene un atributo estático del tipo Comando[ ] donde almacena los objetos correspondientes a los comandos de esta práctica. Por ejemplo, uno de sus elementos será new Ayuda(). Como métodos públicos tendrá los siguientes dos métodos estáticos:

```
static public String AyudaComandos();
static public Comando parseaComando(String[ ] cadenas)
```

  El primer método devuelve una cadena mostrando la ayuda, es decir la información sobre lo que hace cada uno de los comandos. Para ello el método recorre el atributo estático de tipo Comando[ ], invocando para cada uno de los objetos allí contenidos al correspondiente método textoAyuda(). El segundo método parsea el array String[ ] para construir el comando que representa, o en caso de no representar ninguno, devolver null. Este método, al igual que el anterior, recorre el atributo estático de tipo Comando[ ], invocando para cada comando a su correspondiente método parsea hasta encontrar el comando que encaja.

- Controlador: Similar al de la Práctica 1, pero ahora el método realizaSimulacion() utiliza el método parseaComandos de la clase ParserComandos, así como el método ejecuta de la clase Comando. Concretamente este método contendrá las siguientes dos líneas:

```
Comando comando = ParserComandos.parsea(words);
if (comando!=null) comando.ejecuta(this.mundo);
```

  El final de la simulación ocurre cuando se teclea el comando Salir. Por ello, la clase Mundo, como detallaremos más abajo, contendrá un método public boolean esSimulacionTerminada(), que permitirá al controlador saber cuando acabar la simulación.

- Celula: Es una clase abstracta, de la que heredan las clases concretas CelulaCompleja y CelulaSimple. Esta clase contiene un atributo privado protected boolean esComestible. que indica si una celula es comestible o no. Observa que las células simples son comestibles, pero las complejas no lo son. Esta clase contiene dos métodos abstractos:

```
public abstract Casilla ejecutaMovimiento(int f, int c, Superficie superficie);
public abstract boolean esComestible();
```

  El primer método realiza el movimiento de una célula colocada en la posición (f,c) de la superficie, y devuelve la casilla a la que se ha movido la célula o null en caso de

que la célula no se pueda mover. La clase Casilla implementa una coordenada (x,y) y tendrá constructora con argumentos así cómo métodos de consulta a las coordenadas. El segundo método tan sólo devuelve el valor del atributo esComestible. Las clases que heredan de ésta deben implementar de forma correcta estos dos métodos. Es posible que además te hagan falta más métodos y atributos en las clases que heredan, para poder implementar correctamente su funcionalidad.

- Mundo: Su implementación es similar a la de la Práctica 1, pero ahora hay que tener en cuenta que tenemos dos clases de células a repartir aleatoriamente en la superficie, en función de constantes que nos indican cuántas células de cada tipo hay que crear. Por otro lado, el método evoluciona() ya no se puede implementar como en la Práctica 1, ya que ahora no se sabe qué tipo de célula contiene cada celda de la superficie. Para realizar la evolución del mundo tendremos que apoyarnos en el atributo superficie que contiene la clase Mundo. En la descripción de la clase Superficie que aparece a continuación podrás ver que existe un nuevo método en la clase Superficie que permite realizar la evolución. Por otro lado el mundo contiene un atributo private boolean simulacionTerminada que controla cuando termina la simulación. Inicialmente el atributo tiene valor True, y su valor se modificará cuando se ejecute el comando Salir. Por lo tanto necesitarás métodos para modificar y consultar este atributo.
- Superficie: Clase muy similar a la de la Práctica 1, con ligeros cambios que tendrás que realizar. Ahora esta clase incorpora, entre otros, un nuevo método public Casilla ejecutaMovimiento(int f, int c), que permite indicar a la célula de la posición (f,c) que evolucione de acuerdo a sus reglas, y que devuelva la casilla a la que se ha movido (o null) en caso de no moverse.

4. 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.