Ejercicios Arrays Avanzados (III)

1. Arrays Avanzados (III)

Ejercicio 23: Un restaurante nos ha encargado una aplicación para colocar a los clientes en sus mesas. En una mesa se pueden sentar de 0 (mesa vacía) a 4 comensales (mesa llena).

El funcionamiento es el siguiente:
Cuando llega un cliente se le pregunta cuántos son. Como el programa no está preparado para colocar a grupos mayores a 4, si un cliente solicita una mesa con mas comensales (pej, 6), el programa dará el mensaje "Lo siento, no admitimos grupos de 6, haga grupos de 4 personas como máximo e intente de nuevo" y volverá a preguntar.

Para cada grupo nuevo que llega, se busca siempre la primera mesa libre (con 0 personas). Si no quedan mesas libres, se busca una donde haya hueco para todo el grupo (por ejemplo, si el grupo es de dos personas, se podrá colocar en mesas donde haya una o dos personas).

Cada vez que se sientan nuevos clientes se debe mostrar el estado de las mesas.
Los grupos no se pueden romper, aunque haya huecos sueltos suficientes.

A tener en cuenta:

- El programa comienza pidiendo el número de mesas que tiene el restaurante.
- Inicialmente, las mesas se cargan con valores aleatorios entre 0 y 4 y mostrará por pantalla como quedan las mesas inicialmente.
- El programa seguirá pidiendo comensales hasta que se introduzca un valor negativo.

Ejemplo de ejecución:

```
//El usuario ha metido un valor de 10

Estado de las mesas: 3 2 0 2 4 1 0 2 1 1

El usuario pide una mesa para 2.
Por favor, Siéntese en la mesa 3

Estado de las mesas: 3 2 2 2 4 1 0 2 1 1
```

MEDAC

```
El usuario pide una mesa para 4
Por favor, Siéntese en la mesa 7

Estado de las mesas: 3 2 2 2 4 1 4 2 1 1

El usuario pide una mesa para 3
Por favor, Siéntese en la mesa 6

Estado de las mesas: 3 2 2 2 4 4 4 2 1 1

El usuario pide una mesa para 4
Lo siento, no queda sitio en el restaurante.

Estado de las mesas: 3 2 2 2 4 1 4 2 1 1
```

Ejercicio 24: Autómata Celular unidimensional

Supongamos una lista de valores binarios que representan la existencia de células vivas (1) o no (0). Esta lista evoluciona a lo largo del tiempo, creándose nuevas células vivas y/o muriendo otras.

Una celda de la lista evolucionará a partir del estado de sus celdas vecinas (la de su izquierda y la de su derecha) y de ella misma en el instante anterior.

Para ello se siguen las siguientes reglas:

- 000 – 0
- 001 – 1
- 010 – 1
- 011 – 0
- 100 – 1
- 101 – 1
- 110 – 0
- 111 – 0

Una regla `IXD – Y` se interpreta de la siguiente forma: *"La celda X rodeada de las celdas I y D, evolucionará al estado Y"*

Pej:

0**11** – 0 : una celda en el estado 1 rodeada de cero ala izquierda y 1 a la derecha, evolucionará a 0 en el siguiente paso.

1**01** – 1 : una celda en el estado 0 con rodeada de unos, evolucionará a 1 en el siguiente paso.

Vamos a crear un script que nos permita estudiar la evolución del autómata celular antes descrito de tamaño TAM (numero entero mayor o igual a 3) Para ello debes crear las siguientes funciones:

- `inicializarAutomata` : llena el autómata de células muertas.
- `mostrarEstado`: imprime el contenido del autómata teniendo en cuenta que mostrará las células vivas con el carácter '*' y las muertas con '.'
- `primeraJugada`: indica a través de código qué celdas tienen inicialmente células vivas.

Haciendo uso de las funciones anteriores, crea un script que pide al usuario el tamaño del autómata (valor entero mayor o igual a 3) y el número de pasos que va a dar el autómata. Muestra por consola el estado del autómata en cada paso que se vaya dando.

Ejemplo de salida:

Para un autómata de tamaño 15, con la celda 8 como única viva en la primera jugada y 7 pasos, tenemos:

```
Paso 0:. . . . . . . * . . . . . . .
Paso 1:. . . . . . * * * . . . . . .
Paso 2:. . . . . * . . . * . . . . .
Paso 3:. . . . * * * . * * * . . . .
Paso 4:. . . * . . . * . . . * . . .
Paso 5:. . * * * . * * * . * * * . .
Paso 6:. * . . . * . . . * . . . * .
```

MEDAC

Ejercicio 25: La Búsqueda del Tesoro

Crea un script que simule el siguiente juego:
En un tablero, NxM la maquina colocará de forma aleatoria varias minas y un tesoro. El usuario intentará averiguar la posición del tesoro indicando la posición del tablero que quiere mirar. Podrá seguir jugando mientras no encuentre una de las minas (muere) o el tesoro (gana).

Para este ejercicio usa un tablero de 4x5 con 3 minas.
Hay que asegurarse que tanto las minas como el tesoro no se colocan en posiciones ya ocupadas.
En cada iteración del juego se pintará el tablero y se preguntará al usuario qué coordenadas quiere mostrar del tablero.

Usa la consola para pintar el tablero de la siguiente forma:

- Si una coordenada aún no ha sido visitada, pinta un *
- Si una coordenada ha sido visitada y no tiene nada, pinta un _
- Si una coordenada ha sido visitada y tiene una mina, pinta una X (y mata al jugador)
- Si una coordenada ha sido visitada y tiene el tesoro, pinta un € (y el jugador gana)

Puedes pedir al usuario las coordenadas como quieras: ambos valores de golpe o de uno en uno. Lo único a tener en cuenta es que, para el usuario, las coordenadas empiezan en 1, no en 0.

Mejora: si hay una mina a una casilla de distancia de la posición que ha descubierto el usuario, muestra un aviso indicando que tenga cuidado (pero no reveles la posición de la mina).