Fundamentos de la programación 1

Grado en Desarrollo de Videojuegos

Examen de convocatoria ordinaria. Curso 24/25

Indicaciones generales:

- Se entregará únicamente el archivo `Program.cs` con el programa pedido.
- Las líneas 1 y 2 serán comentarios de la forma:
  `// Nombre y apellidos`
  `// Laboratorio, puesto`
- **Lee atentamente el enunciado** e implementa el programa tal como se pide, con los métodos, parámetros y requisitos que se especifican. No puede modificarse la representación propuesta, ni alterar los parámetros de los métodos especificados. No se especifica el modo de paso de los parámetros en los métodos (`out`, `ref`, ...), que debe determinar el alumno. Pueden implementarse todos los métodos adicionales que se consideren oportunos, especificando claramente su cometido, parámetros, etc.
- **El programa debe compilar y funcionar correctamente, y estar bien estructurado y comentado**. Se valorarán la claridad, la concisión y la eficiencia.
- La **entrega:** se realizará a través del servidor FTP de los laboratorios.
  Importante: comprobar el archivo entregado en el puesto del profesor.

Vamos a implementar una versión del *juego de Nim para n jugadores*. Se inicia con varios montones de palillos y un grupo de jugadores que juegan por turnos. Cada jugador, en su turno, tiene que retirar una cantidad de palillos de un montón (los que quiera, pero **al menos uno y de un solo montón**). Gana el jugador que retira el último palillo.

Para entender la mecánica del juego, supongamos que tenemos los jugadores *Ana*, *Berto*, *Carla* y *Humano*, que jugarán cíclicamente en este orden. Los tres primeros son simulados por el ordenador y *Humano* corresponde al usuario. Suponemos 5 montones con la siguiente distribución inicial de palillos: $[2, 3, 4, 1, 2]$. El primer turno se elige aleatoriamente; para nuestro ejemplo supondremos que le toca a *Berto*. A continuación mostramos una posible secuencia de juego con cuatro jugadas (en dos columnas):

```
Empieza el juego:
0 ||
1 |||
2 ||||
3 |
4 ||

Berto quita 1 del montón 4
0 ||
1 |||
2 ||||
3 |
4 |
```

```
Carla quita 1 del montón 0
0 |
1 |||
2 ||||
3 |
4 |

Humano, tu turno:
Montón (-1 para terminar): 1
Palillos: 2

Humano quita 2 del montón 1
0 |
1 |
2 ||||
3 |
4 |
```

En cada estado del juego se muestra la configuración de los montones: una línea por cada montón (numeradas de 0 en adelante), con tantos '|' como palillos tiene el montón (nótese que el primer estado mostrado corresponde al estado inicial [2,3,4,1,2] mencionado arriba). Empieza jugando *Berto*, que quita 1 palillo del montón 4, obteniendo el estado [2,3,4,1,1]. Después juega *Carla*, que quita 1 del

montón 0. Estos dos jugadores son simulados por la máquina, que elige aleatoriamente el montón y los palillos a retirar, como veremos. Después tiene el turno *Humano*, correspondiente al usuario y se solicita la jugada por teclado: en este caso elige retirar 2 palillos del montón 1. Después jugaría *Ana*, luego *Berto*, etc., cíclicamente hasta terminar el juego.

El programa tendrá el siguiente aspecto:

```
using System;
using System.IO;
namespace Main {
  class MainClass {
    static Random rnd = new Random(); // generador de aleatorios
    const int NUM_MONTONES = 5, MAX_PALILLOS = 4;

    public static void Main () {
      string [] jugadores = {"Ana", "Berto", "Carla", "Humano"};
      int [] montones = new int[NUM_MONTONES];
      int turno;
      ...
```

La constante `NUM_MONTONES` representa el número de montones y `MAX_PALILLOS` el número máximo de palillos que puede haber en cada montón. En el método `Main` se define el array `jugadores` con el nombre de los jugadores (se han puesto los nombres del ejemplo inicial, pero podría contener cualesquiera otros). El array `montones` guardará el número de palillos de cada montón. La variable `turno` indica el jugador al que le toca jugar en cada estado del juego (índice entre 0 y 3, correspondiente al vector `jugadores`).

Para desarrollar el juego, se pide implementar los siguientes métodos (se recomienda seguir este orden):

- **[1 pt]** `void Inicializa(int [] montones, string [] jugadores, int turno)`: rellena cada componente del array `montones` con un número aleatorio de palillos entre 1 y `MAX_PALILLOS`; inicializa `turno` con un valor aleatorio entre 0 y el número de jugadores. Recordemos que `rnd.Next(a,b)` genera un aleatorio del intervalo `[a,b)` y que el tamaño de un array `v` puede obtenerse con `v.Length`.

- **[1 pt]** `void Render(int [] montones, string [] jugadores, int turno, int num, int mon)`: `num` representa el número de palillos retirados en la jugada anterior y `mon` el montón del que se han retirado. Si `num=0` (no se han retirado palillos en la jugada anterior) muestra el mensaje `"Empieza el juego"`; si `num>0` (se han quitado palillos en la jugada anterior) muestra un mensaje informando de qué jugador ha quitado palillos de un montón (por ejemplo: `"Berto quita 1 del montón 4"`).

  A continuación, y con independencia del mensaje anterior, muestra el estado de los montones tal como se presenta en el ejemplo de arriba (una línea por montón, con tantos '|' como palillos tiene dicho montón).

- **[1 pt]** `void JuegaHumano(int [] montones, int mon, int pals)`: escribe el mensaje `"Humano, tu turno"` y solicita al usuario una jugada:

  - En primer lugar solicita el montón `mon`. El usuario puede indicar `-1` para terminar el juego (tal como se muestra en el ejemplo), en cuyo caso devuelve este valor en `mon` y termina el método.
  - Si `mon!=-1`, solicitará también un número positivo de palillos `pals` y comprobará que la jugada es válida: el montón existe y tiene al menos `pals` palillos. En ese caso lleva a cabo la jugada, retirando los palillos indicados y devuelve los valores `mon` y `pals`. Si la jugada no es válida, vuelve a solicitarla.

- [**1 pt**] `void JuegaMaquina(int [] montones, int mon, int pals)`: genera aleatoriamente un montón `mon` y un número positivo de palillos `pals`, tales que el montón existe y tiene al menos `pals` palillos; lleva a cabo la jugada, retirando los palillos indicados y devuelve los valores `mon` y `pals`.

- [**1 pt**] `bool FinJuego(int [] montones)`: devuelve `true` si todos los montones están vacíos; `false` en caso contrario.

Con estos métodos ya podemos implementar una primera versión del método `Main`: inicializa el estado del juego, lo muestra en pantalla e implementa el bucle principal. Este bucle dará el turno cíclicamente a cada uno de los jugadores hasta que el juego llegue a su fin o el usuario decida terminarlo (seleccionando el montón `-1` tal como se ha explicado). Todos los jugadores utilizan el método `JuegaMaquina`, excepto `"Humano"`, que utiliza `JuegaHumano`. Cada vuelta del bucle renderiza el estado del juego tras la jugada, tal como se muestra en el ejemplo, y actualiza el turno. Al final del bucle el programa indicará quién es el ganador.

A continuación vamos a implementar otros dos métodos para salvar y recuperar la partida:

- [**1 pt**] `void GuardaPartida(int [] montones, int turno)`: guarda el estado actual de la partida (contenido de los `montones` y `turno`) en el archivo `"saved"`. En ese archivo escribirá una única línea de enteros con el contendido de los montones y al final el turno. En el ejemplo del principio, el estado inicial se guardaría como: `2 3 4 1 2 1` (la distribución inicial de palillos y un `1` al final que indica que es el turno de `Berto`).

- [**1 pt**] `void LeeArchivo(int [] montones, int turno)`: lee del archivo `"saved"` (en el formato anterior) el estado del juego, y lo devuelve en los parámetros `montones` y `turno`. Recordemos que dada una cadena de texto `s`, la llamada `s.Split(' ')` devuelve un array de strings utilizando `' '` como separador.

Vamos a enriquecer el juego añadiendo una *regla de salto de turno*: cuando un jugador al retirar palillos consigue un vector palíndromo repite turno. Un vector es palíndromo si se lee igual de izquierda a derecha que de derecha a izquierda. Por ejemplo, el vector `[1,2,3,2,1]` es palíndromo, así como `[5,6,6,5]`. Implementaremos el método:

- [**1 pt**] `bool Palindromo(int [] montones)`: determina si el vector montones es palíndromo de acuerdo a la definición anterior.

[**2 pt**] Una vez implementados estos métodos, extender el método `Main` de modo que:

- Al principio pregunte al usuario si quiere recuperar una partida salvada o comenzar con una aleatoria. En el primer caso inicializará el estado con el método `LeeArchivo` y en segundo con `Inicializa`.

- Después entrará en el bucle principal, como antes, pero incorporando la regla de salto de turno.

- Si el usuario aborta la partida (indicando el montón `-1` en `JuegaHumano`), preguntará si se quiere guardar la partida. En ese caso la guardará con el método `GuardaPartida`.