Fundamentos de la programación 1

Grado en Desarrollo de Videojuegos

Convocatoria extraordinaria de junio. Curso 23/24

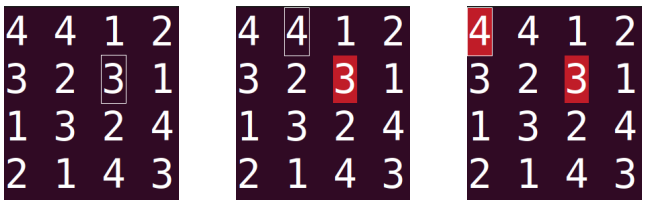
Indicaciones generales:

- Se entregará únicamente el archivo `Program.cs` con el programa pedido. Se proporciona una **plantilla** con el esquema del programa y algunos métodos ya implementados.
- Las líneas 1 y 2 serán comentarios de la forma:
  `// Nombre 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 el Hitori (Nikoli, 1990), juego que se desarrolla en un **tablero cuadrado**. Al principio todas las casillas contienen números del 1 al 9, y se trata de ir tachando algunos de esos números hasta conseguir:

- Que no haya dos números duplicados en ninguna fila ni columna.
- Que no haya dos casillas contiguas tachadas en horizontal ni en vertical (se admiten en diagonal).

A continuación se muestra un ejemplo de tamaño 4x4 con tres estados de juego:



A la izquierda se muestra el tablero inicial, con el cursor en la posición (1,2). En el centro se muestra en estado parcialmente resuelto (se ha tachado el 3 de fondo rojo). El último tablero muestra el juego resuelto: no hay repeticiones por fila ni columna, ni casillas contiguas tachadas.

Nuestra implementación mostrará el tablero en pantalla y la posición actual del cursor, que podrá moverse con las teclas de desplazamiento. Con la barra de espacio se marcará como tachado el número de la posición actual, o se desmarcará si está tachado.

En la plantilla (`Program.cs`) se incluyen las variables que representan el juego y una inicialización correspondiente al tablero de ejemplo (el programa debe desarrollarse para poder jugar con tableros de cualquier tamaño $N \times N$, no solo 4×4):

```
class Hitori{
  static void Main() {
    int  [,] tab;      // números del tablero
    bool [,] tachadas; // casillas tachadas; true: tachada
    int fil, col;      // posición del cursor
```

```
// inicialización
fil = col = 0;
tab = new int [,]{
    {4, 4, 1, 2},
    {3, 2, 3, 1},
    {1, 3, 2, 4},
    {2, 1, 4, 3}};
tachadas = new bool [,]{
    {false, false, false, false},
    {false, false, false, false},
    {false, false, false, false},
    {false, false, false, false}};
```

El estado interno del juego queda representado con las variables `tab`, `tachadas`, `fil` y `col`. La variable `tab` es un matriz de enteros con los valores del tablero y `tachadas` es una matriz de booleanos del mismo tamaño, que lleva cuenta de las casillas que ha tachado el jugador (en este ejemplo, todas las componentes a `false`, sin tachar). Las variables `fil,col` almacenan la posición actual del cursor.

Implementaremos los siguiente métodos (se recomienda seguir el orden propuesto, comprobando el funcionamiento de cada uno antes de pasar al siguiente):

- [**1**] `void Render(int [,] tab, bool[,] tachadas, int fil, int col)`: muestra en pantalla el estado del juego. Para proporcionar el aspecto **cada casilla debe ocupar dos espacios en pantalla** y los números tachados se mostrarán con fondo rojo (`Console.BackgroundColor = ConsoleColor.DarkRed`), tal como se aprecia en el ejemplo.

  Para situar el cursor basta con llamar al método `Console.SetCursorPosition(left,top)` con los parámetros adecuados.

- [**1**] `void ClickCasilla(bool [,] tachadas, int fil, int col)`: tacha la casilla de la posición actual `(fil,col)` si no está tachada y no hay ninguna ya tachada en las posiciones contiguas (en horizontal o en vertical). Si la casilla de dicha posición está tachada, pasa a estar no tachada.

  Nota: este método solo afecta al estado interno del juego. No tiene ningún efecto visible en pantalla.

- [**0.5**] `void ProcesaInput(char c, bool[,] tachadas, int fil, int col)`: procesa el input codificado en `c`:

  - `'u','d','l','r'` (up, down, left, right): actualiza la posición actual `(fil,col)` controlando que no se salga de los límites de la cuadrícula (no hace nada si se sale).
  - `'c'`: marca o desmarca como tachada la posición `(fil,col)` utilizando el método anterior `ClickCasilla`.

El método `LeeInput()` se da implementado en la plantilla. Para las teclas de cursor y devuelve los caracteres correspondientes `'u','d','l','r'` correspondientes. Para el espacio devuelve `'c'` y para la tecla *Escape* devuelve `'q'`.

Utilizando los métodos anteriores ya puede implementarse una primera versión del método `Main` con el bucle principal, que debe (por este orden): leer input, procesarlo y mostrar en pantalla el tablero resultante. El bucle terminará cuando el usuario pulse *Escape*.

A continuación, implementaremos algunos métodos para comprobar que no haya repetición de números en las filas ni las columnas.

- [**1**] `int [] DameFil(int[,] tab, bool [,] tachadas, int i)`: devuelve un array con los números de fila i-ésima de la matriz `tab`; para los que estén tachados devolverá -1. Por ejemplo, para el tablero resuelto del ejemplo inicial (el de la derecha), para la fila 1 se obtendrá el vector `[3,2,-1,1]`. Análogamente, implementar el método:

  `int [] DameCol(int[,] tab, bool [,] tachadas, int i)`: devuelve los número de la columna i-ésima de `tab`; -1 para lso tachados.

- [**1.5**] `bool RepetidosVector(int[] v)`: determina si el vector `v` contiene algún número repetido.

- [**1.5**] `bool RepetidosMatriz(int[,] tab, bool [,] tachadas)`: utilizando los métodos anteriores, determina si la matriz `tab` contiene alguna fila o columna con números repetidos.

Utilizando estos métodos puede extenderse el método `Main` para detectar la terminación del juego cuando el tablero no contenga repeticiones en las filas ni en las columnas.

A continuación, implementaremos métodos para leer y guardar en archivos:

- [**1**] `void LeeArchivo(string file, int[,] tab, bool[,] tachadas, int fil, int col)`: lee el estado de juego del archivo `file` y lo devuelve en los parámetros `tab` y `tachadas`. El cursor `(fil,col)` se ubicará en `(0,0)`. El archivo de entrada tiene el siguiente formato:

```
4
4 4 1 2
3 2 3 1
1 3 2 4
2 1 4 3
f f f f
f f t f
f f f f
f f f f
```

  Este archivo corresponde al tablero central del ejemplo inicial (parcialmente resuelto). La primera línea indica el tamaño del tablero (4 en este caso); las siguientes líneas contienen los números del tablero (4 líneas, con 4 números cada una, en este caso). Y las siguientes indican si el correspondiente número está tachado (`'t'`) o no (`'f'`). En este caso solo está tachado un 3 en la segunda fila. Puede asumirse que el archivo tiene un formato correcto (no hay que tratar las excepciones).

  Para implementar este método será util trocear cada línea utilizando el blanco como separador:

```
string [] nums = (f.ReadLine()).Split(" ",StringSplitOptions.RemoveEmptyEntries);
```

  En el archivo `ejemplo` se proporciona otro tablero de prueba.

- [**1**] `void void SalvaArchivo(string file, int[,] tab, bool[,] tachadas)`: guarda en el archivo `file` las matrices `tab` y `tachadas` en el formato anterior.

[**1.5**] Por último, se extenderá el método `Main` para que:

- Al arrancar el programa permita utilizar el tablero de ejemplo o cargar uno de un archivo que se solicitará al jugador.

- Cuando el jugador aborte la ejecución (con *Escape*) permita guardar el estado del juego en un archivo solicitado al jugador.