Fundamentos de la programación

Grado en Desarrollo de Videojuegos

Práctica 2: Kakurasu

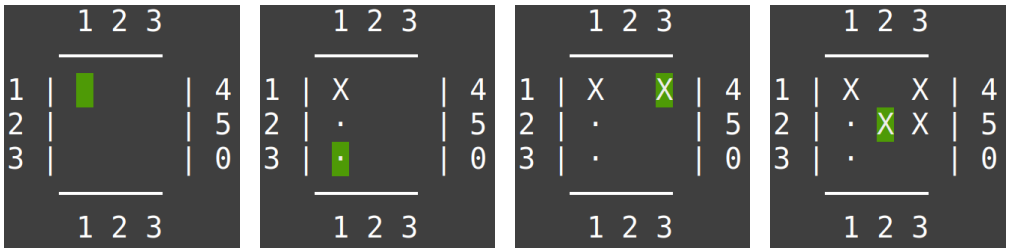
Indicaciones generales:

- La línea 1 del programa y siguientes deben contener los nombres de los alumnos de la forma (una línea por alumno):

  ```
  // Nombre Apellido1 Apellido2
  ```

- **Lee atentamente** el enunciado y desarrolla el programa tal como se pide, con la representación y esquema propuestos, implementando los métodos que se especifican, **respetando los parámetros y tipos de los mismos**. Pueden implementarse los métodos auxiliares que se consideren oportunos comentando su cometido, parámetros, etc.
- El programa, además de correcto, debe estar bien estructurado y comentado. Se valorarán la claridad, la concisión y la eficiencia.
- La entrega se realizará a través del campus virtual, subiendo únicamente el archivo `Program.cs`, con el programa completo.
- El **plazo de entrega** finaliza el 2 de diciembre.

Vamos a implementar el **Kakurasu**, un puzzle que se juega en un tablero $N \times N$ (con $N \leq 9$), en el que hay que marcar casillas para que la suma de sus posiciones coincida con los números indicados para cada fila y cada columna. Para calcular la suma de una fila, se suman las posiciones de las casillas marcadas en dicha fila mientras avanzamos en ella de izquierda a derecha: la primera cuenta 1, la segunda cuenta 2, etc. Para la suma de una columna, se suman las posiciones de las casillas marcadas en la columna avanzando de arriba a abajo: la primera 1, la segunda 2, etc. Por ejemplo, para $N = 3$, podemos tener la siguiente secuencia de juego:



A la izquierda se muestra el estado inicial, sin resolver. El tablero de juego está delimitado por las líneas verticales y horizontales. La fila superior y la columna izquierda (con 1,2,3) son simplemente la numeración de las filas y columnas, para facilitar al jugador la identificación de las posiciones. La columna a la derecha (4,5,0) determina la *suma objetivo* de las filas y la fila inferior (1,2,3) es la *suma objetivo* para las columnas. En este caso, las posiciones marcadas de la primera fila deben sumar 4, las de la segunda 5 y las de la tercera 0; las posiciones marcadas de las columnas deben sumar 1, 2 y 3, respectivamente. Las casillas marcadas se representan con `'X'` y la posición de la casilla activa se muestra en verde.

El jugador podrá moverse por la cuadrícula con las flechas de cursor, marcar casilla con `'x'` y borrar con la barra espaciadora. También puede anotar casillas con `'v'` para recordar que no deben ser marcadas, aunque no es obligatorio (aparecen con `'·'` en pantalla), y podrá terminar el juego con la tecla de *Escape*.

El segundo tablero del ejemplo corresponde a un estado intermedio en el que se ha marcado la esquina superior izquierda. Como la suma objetivo de esa columna es 1, dicha columna no puede contener más marcadas y esa columna se ha completado. Se han anotado el resto de posiciones con `'·'`, para recordar que no deben ser marcadas. En el tercer tablero hemos marcado la última posición de la fila 1, de modo que las posiciones marcadas en esa fila (la 1 y la 3), alcanzan la suma objetivo, 4. En el último tablero hemos marcado otras dos posiciones en la segunda fila, de modo que el juego está terminado. Nótese que no es necesario anotar con `'·'` las casillas restantes para comprobar la

terminación del juego.

Para desarrollar nuestra implementación, el estado del juego viene dado por las siguientes variables:

- `char [,] tab`: matriz de caracteres (de tamaño $N \times N$) que codifica el estado de las casillas: `'X'` denota casilla marcada, `'·'` anotada y `' '` libre.
- `int [] objetivosFila, objetivosColumna`: arrays (de tamaño `N`) con las sumas objetivo de las filas y las columnas respectivamente.
- `int fil, col`: posición activa en el tablero (fila y columna donde se puede marcar/desmarcar casilla).

Implementaremos los siguientes métodos (por este orden):

- `void Inicializa(int [,] mat, char [,] tab, int[] obFil, int [] obCol,`
  `int fil, int col)`: recibe una matriz de enteros `mat` que representa la matriz de juego y los objetivos por fila y columna, y construye la representación descrita arriba (con matriz de caracteres). Es decir, debe inicializar la matriz `tab` y los arrays de sumas objetivo `objFil` y `objCol`, así como la posición de la casilla activa `fil`, `col` en `(0,0)`.

  A continuación, se muestra un ejemplo de matriz `mat` y el tablero `tab` que debe generar, así como las filas y columnas objetivo, y la posición inicial de la casilla activa:

```
int [,] mat =                char [,] tab =          obFil = {1,2,3}
 {{1,0,1,4},                  {{'X',' ','X'},        obCol = {4,5,0}
  {2,0,0,5},                   {'·',' ',' '},        fil = col = 0
  {2,0,0,0},                   {'·',' ',' '}}
  {1,2,3,0}}
```

  Nótese que la matriz `mat` tiene tamaño $(N+1) \times (N+1)$, de modo que la última fila y columna corresponden las sumas objetivo para las filas y las

columnas respectivamente (el 0 de la esquina inferior derecha es irrelevante). Los valores `0` representan casilla libre, `1` negra y `2` anotada.

- `void Render(char[,] tab, int[] objFil, int [] objCol, int fil, int col)`:
  muestra la cuadrícula en pantalla en el formato descrito, **con la numeración de filas y columnas y las líneas delimitadoras**, asumiendo que $N \leq 9$. Nótese, que las casillas se muestran con un espacio por delante para mejorar el aspecto visual.

  Este método debe mostrar primero el tablero completo escribiendo de arriba abajo y de izquierda a derecha, **sin tener en cuenta la casilla activa**. Después, para situar esa casilla utilizaremos `Console.SetCursorPosition(left,top)` y escribiremos el carácter correspondiente con los colores adecuados (`Console.ForegroundColor = ConsoleColor.White`, `Console.BackgroundColor = ConsoleColor.Green`).

En la plantilla `Program.cs` se incluye una matriz `ex1` con el tablero que hemos utilizado como ejemplo. Desde el método `Main` ya se puede invocar a `Inicializa` y `Render` para mostrarlo en pantalla y comprobar el buen funcionamiento de estos métodos.

En la plantilla también se incluye el método `LeeInput` para leer el input de usuario y codificarlo en caracteres simples (véase esta codificación en el propio método). A continuación implementaremos:

- `void ProcesaInput(char tecla, int fil, int col, char[,] tab)`:
  procesa el input codificado en `tecla`:

  - `'u','d','l','r'` (up, down, left, right): actualiza la posición de la casilla activa `(fil, col)` según la dirección indicada. Si dicha posición se sale por la derecha debe aparecer por la izquierda, y viceversa; análogo si se sale por arriba o por debajo.

- `'x','b','s'` (negro, anotada, limpiar casilla): coloca en `mat` el valor correspondiente, en la posición de la casilla activa (nótese que `LeeInput` devuelve `'s'` para la barra espaciadora).

Con este método ya se puede implementar una primera versión del bucle principal, que permitirá mover la posición de la casilla activa y rellenar casillas. Los siguientes métodos servirán para comprobar la terminación del juego, cuando las casillas marcadas verifiquen las condiciones requeridas.

- `int SumaFil(char[,] tab, int fil)`: calcula la suma de las posiciones marcadas de la fila `fil`.

  `int SumaCol(char[,] tab, int col)`: calcula la suma de posiciones marcadas de la columna `col`.

- `bool Terminado(char[,] tab, int[] objFil, int[] objCol)`: utilizando los métodos anteriores, devuelve `true` si todas las filas y columnas verifican las reglas del juego (suma de posiciones marcadas igual a la suma objetivo por filas y por columnas), o `false` en caso contrario. Este método se utilizará en el bucle principal del `Main` para determinar si el juego ha terminado.

Con estos métodos ya se puede tener una versión funcional del juego, completando el bucle principal (en `Main`), que terminará cuando el juego esté completado o el usuario aborte la partida (con *Escape*).

A continuación haremos algunas extensiones:

- `(bool [], bool []) Incorrectas(char[,] tab, int[] objFil, int[] objCol)`:
  utilizando los métodos `SumaFil` y `SumaCol` determina las filas y columnas incorrectas del tablero y devuelve el resultado en los arrays `filsInc` y

`coslInc`. La posición `i` de `filsInc` será `true` si la fila `i`-ésima es incorrecta y `false` en caso contrario, y análogo para `colsInc`.

- `void RenderIncorrectas(char[,] tab, int[] objFil, int[] objCol, int fil, int col, bool[] filsInc, bool[] colsInc)`: renderiza el tablero con el metodo `Render` y después, muestra por debajo los índices de las filas y las columnas incorrectas.

El programa principal debe, en primer lugar, inicializar el juego con el tablero de ejemplo proporcionado y renderiza el estado inicial. Después, en cada iteración del bucle principal, debe solicitar el input del usuario (`LeeInput`) y procesarlo. Si el usuario solicita abortar el juego, terminará el bucle. En otro caso, debe procesar el input (`ProcesaInput`) y renderizar el estado actual (`Render`). Además, si el usuario ha solicitado pista, renderizará las filas y columnas incorrectas.