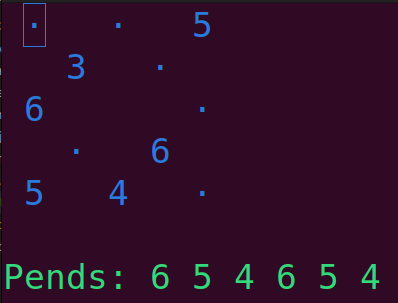
**Fundamentos de la Programación 2** Grado en Desarrollo de Videojuegos **Convocatoria ordinaria. Curso 22/23**

Vamos a implementar una versión simplificada del juego *PuzLogic*1, un juego de tablero que tiene el siguiente aspecto:



A la izquierda se muestra un tablero en su estado inicial. Los números azules vienen fijados y no pueden alterarse durante el juego. Los puntos azules representan huecos vacíos que hay que rellenar con los números de la lista de pendientes, *Pends*, que aparece abajo en verde (puede contener repeticiones y el orden no es relevante). El juego consiste en colocar todos los números de *Pends* en los huecos vacíos de modo que **no se repitan números en ninguna fila, ni columna**. Se asume que todos los números están en el intervalo [1*,* 9] y que la cantidad de huecos coincide con la de números pendientes de colocar.

En la imagen de la derecha se muestra el tablero tras haber colocado dos números (6 y 4), que aparecen en amarillo y se han eliminado de la lista de pendientes. Los números que coloca el jugador no son fijos y pueden quitarse y devolverse a la lista *Pends*. Nótese que hay también casillas muertas que no se utilizan (como la que hay entre el 5 y el 4 en la última fila).

Inicialmente el cursor está en la esquina superior izquierda (posición (0,0)) tal como aparece

en el primer tablero. El jugador podrá moverse por la cuadrícula con las flechas de cursor (en el segundo tablero el cursor está en la posición (1,3)), colocar dígitos pulsando las teclas numéricas y quitar dígitos (amarillos) con la tecla ’s’.

1Edurardo Barreto. Online en: [https://www.coolmathgames.com/0-](http://www.coolmathgames.com/0-puzlogic)puzlogic

Para representar el tablero utilizaremos la clase Tablero con los siguientes atributos (véase archivo Tablero.cs):

, ,

**class** Tablero {

**int** [,] tab; // matriz de números

**bool** [,] fijas; // matriz de posiciones fijas Lista pend; // lista de dígitos pendientes

**int** fil,col; // posición del cursor (fila y columna)

z r

En la matriz tab cada posición contiene un dígito con el siguiente significado:

-1: casilla muerta.

0: hueco vacío que el jugador puede rellenar. 1..9: representa el propio valor.

~~La matriz fijas es de las mismas dimensiones que tab.~~ ~~Cada componente será false si corres- ponde a un hueco en tab inicialmente (será una casilla no fija) y será true en otro caso (casilla fija o muerta)~~. El campo pend es la lista de números pendientes y las coordenadas (fil,col) corresponden a la posición del cursor en el tablero.

~~Para la clase Tablero implementaremos los siguientes métodos, todos públicos salvo que se indique lo contrario:~~

**~~[1]~~** ~~Tablero(int [,] tb, int [] pd) (constructora): crea las matrices tab y fijas con las dimensiones de tb; inicializa tab con los valores de tb y fijas tal como se ha explicado arriba. Crea la lista pend e inserta los números dados en pd. Por último situa la posición del cursor en (0,0).~~

~~Para inicializar el juego, desde el método Main llamaremos a esta constructora con las variables tabEj y pendEj, que contienen el tablero el ejemplo inicial.~~

**~~[1]~~** ~~void Render(): dibuja en pantalla el estado actual del juego tal como aparece en los ejemplos,~~ **~~con las casillas separadas por un blanco en horizontal~~** ~~y el cursor situado en la posición correspondiente.~~

Se puede limpiar pantalla con Console.Clear();, cambiar el color del texto a azul con Console.ForegroundColor=ConsoleColor.Blue; (análogo con Yellow o Green); el mé- todo Console.SetCursorPosition(left,top); permite situar el cursor en la posición indicada (la posición (0,0) corresponde a la esquina superior izquierda). ~~Para escribir la lista de pendientes recordemos que la clase Lista incluye el método (sobrecargado) ToString, es decir, se puede escribir directamente con Console.Write(...).~~

**[0.5]** void MueveCursor(char c): modifica las coordenadas (fil,col) en función del valor de c: ’l’ izquierda, ’r’ derecha, ’u’ arriba, ’d’ abajo, siempre que el movimiento no saque el cursor del tablero (en otro caso, no hace nada).

**[1.5]** private bool NumViable(int num): comprueba si el dígito num puede colocarse en la componente (fil,col) de tab, i.e., si en esa componente hay un hueco libre y además ni la fila fil, ni la columna col contienen ya ese dígito.

**[1]** bool PonNumero(int num): pone el dígito num en la componente (fil,col) del ta- blero si es viable (de acuerdo al método anterior) y aparece en la lista pend. En ese caso, elimina el dígito de pend y devuelve true; en otro caso devuelve false.

1. bool QuitaNumero(): si la componente (fil,col) del tablero tiene un dígito (*>* 0) y no es fijo, lo elimina y lo devuelve a pend. En ese caso devuelve true; en otro caso false.

**[0.5]** bool FinJuego(): determina si el juego ha terminado, i.e., si se han puesto todos los números.

El archivo Program.cs incluye el método LeeInput ya implementado, incluyendo la lectura de dígitos aislados. Se pide implementar los siguientes métodos en esa misma clase:

**[0.5]** void ProcesaInput(Tablero tab, char c): determina la acción a realizar en fun- ción de c:

* + Si es uno de los caracteres ’l’,’r’,’u’,’d’ cambia la posición del cursor en el tablero con el método MueveCursor.
  + Si es el carácter ’s’ llama a QuitaNumero para quitar del tablero el dígito de la posición actual.
  + Si corresponde a un dígito 1-9, lo convierte a entero y llama al método PonNumero para colocarlo en el tablero. Recordemos que la conversión de char a int puede hacerse con int val = (int) (c-’0’).

**[1]** void LeeNivel(string file, int [,] tb, int [] pd): lee del archivo file un tablero de juego y un array de números pendientes y los devuelve en los parámetros tb y pd.

En el archivo ex.txt se proporciona un archivo de ejemplo (correspondiente al tablero que hemos utilizado de ejemplo). La primera línea contiene el número de filas y columnas del tablero. Cada una de las siguientes líneas contiene una fila del tablero codificada con enteros con el mismo significado explicado el principio. La última línea contiene la lista de números pendientes.

Para facilitar el procesamiento, cada línea puede descomponerse directamente en subca- denas utilizando el espacio en blanco como separador:

string [] digs = f.ReadLine().Trim().Split(” ”,StringSplitOptions.RemoveEmptyEntries)

Por ejemplo, para la segunda línea del archivo de ejemplo, esta instrucción devolverá en

digs los strings "0", 1", "0", 1", "5".

**[1 pt]** void Main(): inicializa el tablero de juego e implementa el bucle principal, utilizan- do los métodos anteriores. En cada iteración debe recoger el input del jugador, procesarlo y dibujar el estado del juego. El juego terminará cuando se detecte el final del juego (método FinJuego) o cuando el usuario decida abandonar con ’q’ (ya considerado en el método LeeInput).

Al arrancar el programa este método debe permitir que el usuario elija si quiere jugar con el tablero de ejemplo proporcionado en la plantilla o leerlo del archivo (ex.txt).

Por último, extenderemos la clase Tablero con un nuevo método que debe implementarse de manera **recursiva**:

**[1]** Lista PosiblesRec(): devuelve una lista con los valores del intervalo [1*,* 9] que pue- den colocarse en la posición actual (fil,col) del tablero, es decir, que están en la lista de pendientes y son *viables* (de acuerdo al método NumViable). **Nota**: no puede utilizarse ningún bucle en esté método ni los auxiliares que pudiera requerir.

Extender la funcionalidad de ProcesaInput para que muestre en pantalla los números posibles de la posición actual del tablero con el input ’p’.