Hidato - Haskell

Hidato es un juego de lógica creado por el Dr. Gyora Benedek, un matemático israelí. El objetivo de Hidato es rellenar el tablero con números consecutivos que se conectan horizontal, vertical o diagonalmente.

Estudiantes:
- Emmanuel Torres González
- Raúl Beltrán Gómez

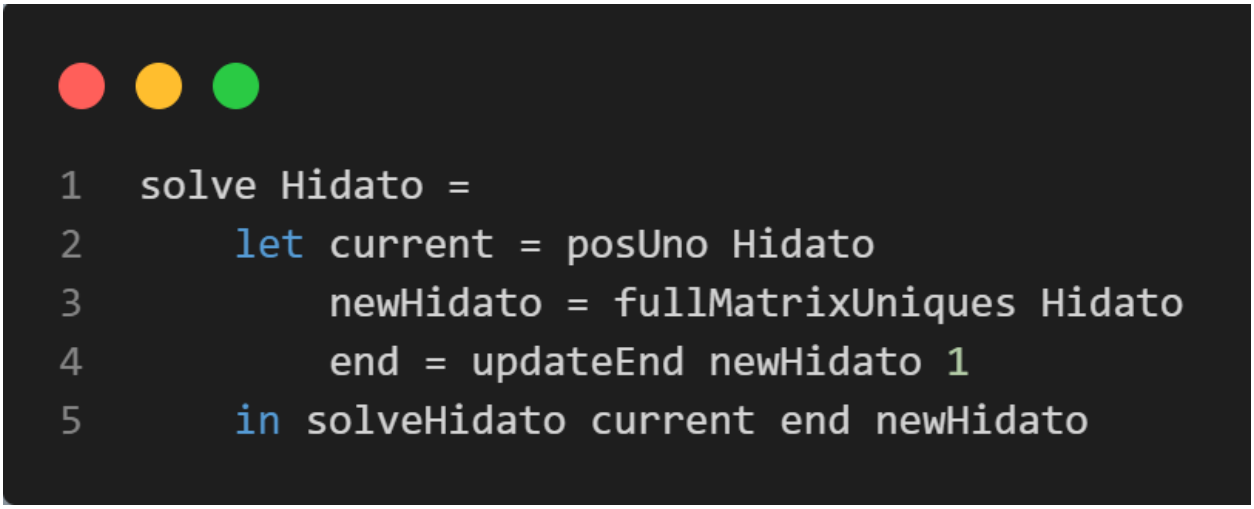
Solucionador

Existe una función principal solve encargada de pasar toda la información necesaria al solucionador, solveHidato.

- **current**: Es la posición del 1. Resultado de la función **posUno**
- **end**: Es el menor número de los mayores que 1 que existen en el Hidato.
- **newHidato**: Es un Hidato ya casi resuelto, con las posiciones obvias. Resultado de la función **fullMatrixUniques**, en el Hidato a simple vista pueden determinarse casillas en las que solo puede ir número y ese es el trabajo de esa función. Ejemplo

		4			
	x			5	6
2				8	x

x = 3 x = 9



```
solve Hidato =
    let current = posUno Hidato
        newHidato = fullMatrixUniques Hidato
        end = updateEnd newHidato 1
    in solveHidato current end newHidato
```

El solucionador es **solveHidato**. Este recibe la posición inicial y la final sobre la cual debe encontrar un recorrido en el Hidato.
Al inicial en el current empieza a expandirse por su adyacentes y diagonales (8 vecinos) sumando en el que este disponible, luego empieza a expandirse a través del escogido hasta encontrar el end sino regresa hacia atrás para escoger otro vecino por el cual moverse.

- **index**: Esta función devuelve el valor de una coordenada en el Hidato
- **tLoop**: Esta función ciclo, la cual recibe una función a ejecutar durante el ciclo no finalice.

```haskell
solveHidato :: [Int] -> [Int] -> [[Int]] ->(Bool, [[Int]])
solveHidato current end Hidato =
    -- Comprueba que haya llegado al objetivo por el camino correcto
    -- sino regresa hacia atrás y se mueve por otra vecino de la casilla
    if index Hidato current  == index Hidato end
        then if current == end
                then
                    let next_end = updateEnd Hidato ((index Hidato end)+1)
                    in if next_end == []
                        then (True, Hidato)
                        else solveHidato end next_end Hidato

                else (False, [])
        else let neigbors = validNeigbors current Hidato
             in tLoop fun Hidato (False, []) neigbors end ((index Hidato current)+1)
                where
                    fun Hidato acc neigbor end value  =
                            let newHidato = changeMatrix Hidato value neigbor
                            in  solveHidato neigbor end newHidato

```

Generador

Método de generación:

1. Se genera un tablero valido con una semilla dada, la semilla consta de una matriz de `0` (espacios vacíos) y `1` (Obstáculos), además contiene el numero mayor y menor, donde el número menor debe ser 1 y el mayor la cantidad de espacios accesibles de la matriz, este tablero válido se genera con el algoritmo de solucionar (el cual es un backtracking). Es precisamente este paso del algoritmo el más costoso dado que puede tardar en recorrer tableros válidos para encontrar una solución.
2. Una vez esta generado este tablero se procede a eliminar casillas del mismo de tal forma que al quedar estas en blanco la solución sea única, si la solución es única con esa casilla en blanco significa que podemos dejar la misma en blanco. Esto lo hacemos con un algoritmo que rellena siempre valores únicos en el tablero, en Sudoku Hidato hay valores que son únicos (obligatorios) en tableros dados, y nuestro algoritmo detecta muchos de estos valores que deben ser únicos, realizamos el paso poner en blanco una casilla y rellenar valores únicos hasta que pasemos por todas las casillas del tablero. Se asegura que si el algoritmo de llenar valores únicos le da valor a esa casilla entonces esa casilla sera única para el tablero que se analiza, por

lo tanto la solución sera única si dejamos la misma en blanco.
Ejemplo:
[ ][7][1] En esta matriz nuestro algoritmo decide que entre el `3` y el `1` debe ir un `2`, luego la casilla [ ][ ][ ] del `2` puede estar vacía xq la solución es única, cuando pasa por la casilla entre en `3` y el `5` [3][ ][5] decide que debe ir un `4` xq en la de encima se relleno con `2` en el recorrido del algoritmo, en la casilla encima del `5` debe ir un `6`, xq la conexión entre `5` y `7` solo puede ser entonces por dicha casilla... nuestro algoritmo se asegura de llenar casillas que solo tienen valores únicos(obligatorios), haciendo esto sobre el algoritmo explicado(eliminar casillas y probar solución) se asegura que cada matriz que vaya quedando en cada paso tendrá solución única xq en cada espacio en blanco solo puede ir una solución.

Nótese que en la matrix [ ][7][ ] no puede colocar el 2 en primer momento, de hecho esta matrix no tiene [ ][ ][1] solución en un primer momento se genera el `6` en medio, el cual es [3][ ][5] obligatorio, luego se genera entre en `3` y el `5` el número `2`, para conectar al `1` con el `3`, si el 7 no existiera nuestro algoritmo no haría cambios.

Algoritmos:

1. Detección de valores únicos:
   Este algoritmo detecta si un espacio en blanco debe tener un valor único, esto esta dado por algunas condiciones que puede presentar el tablero, por ejemplo, entre dos números con dos unidades de diferencia debe existir el de en medio de ambos para conectarlos, si entre ellos dos solo existe una casilla entonces esta casilla sera el valor intermedio, nuestro algoritmo pregunta si el valor en blanco que estamos analizando es el unico valor en medio de dos valores con unidades diferentes y decide si es entonces único o no, también este algoritmo pasa por los valores que no están en blanco y pregunta si solo tienen una casilla vacía adyacente, si esto pasa pregunta si el antecesor y sucesor de este número ya se encuentran en la matriz, si uno de los dos no se encuentra pues entonces el valor vacio adyacente al mismo debe ser el este que no se encuentra en la matriz dado que en otro caso no es alcanzable por ambos lados y no es solución, tiene defido el caso especial para el mayor y el menor número de la matriz. Una vez comprueba todo esto pasa a preguntar por todos los valores admisibles de un espacio vacio, eso se detecta por los adyacentes y por los valores que ya existen en la matriz, en caso de que solo tenga un valor admisible pues ese es su valor.
2. Llenar la matriz de valores únicos: Este algoritmo pasa por cada uno de elementos de la matriz y aplica el algoritmo de detección de de valores únicos para rellenar cada posición visitada o sus adyacentes según el caso.