Documentación Nonogram

Materia: Lógica para ciencias de la computación

Etapa del Proyecto: Etapa 2

Fecha: 10/06

Comisión: 12

Integrante 1: Marcos Simón Fernandez - 142203

Integrante 2: Marcos Goizueta- 142071

ÍNDICE

Resolución de requerimientos funcionales (solo prolog):

Botón “revelar celda”:

Implementación completa en React. No tiene efecto hasta que se genera el tablero resuelto en prolog.

Botón “mostrar solución”:

Implementación completa en React. No tiene efecto hasta que se genera el tablero resuelto en prolog.

Resolución del nonograma en prolog:

Para resolver el nonograma al inicio del juego para así guardarlo en React y que el usuario pueda utilizar las ayudas de **revelar celda** y **mostrar solución** todas las veces que quiera seguimos la siguiente estrategia:

Primero nos fijamos todas las columnas que tengan una única solución mirando el tamaño de las mismas y el de las pistas, y las pintamos. Además las celdas que no van pintadas de estas columnas tendrán cruces, lo que también reducirá las posibilidades de distintas soluciones luego. Con esta nueva grilla que tiene el pintado inicial y estas columnas pintadas, vamos a un nuevo predicado que va formando las soluciones dejando espacios al inicio si no consigue resolver la pista, y para formarlas tiene en cuenta que sean compatibles con las cruces y celdas pintadas de la grilla que le pasamos. A continuación veremos más en detalle cómo se hace todo este proceso.

solveGame/4:

Este predicado cuenta con los siguientes parámetros: solveGame(+RowClues,+ColClues,+Grid,-SolvedGrid). Comienza pintando las columnas que tienen una única solución posible a través de ***paintInitialCols/3.*** Teniendo esta nueva grilla con posiblemente más celdas pintadas, intento resolver el nonograma por medio de ***trySolveGrid/4***.

```
solveGame(RowClues,ColClues,Grid,SolvedGrid):-
transpose(Grid,TransposeGrid),
paintInitialCols(ColClues,TransposeGrid,NewTransposeGrid),
transpose(NewTransposeGrid,NewGrid),
trySolveGrid(RowClues,ColClues,NewGrid,SolvedGrid).
```

paintInitialCols/3:

Este predicado cuenta con los siguientes parámetros: paintInitialCols(+ColClues,+Grid,-NewGrid). A través de ***oneSolve/2*** verifico si la columna tiene una única posible solución teniendo en cuenta las pistas y el tamaño de la misma. Si esto se cumple entonces la resuelvo a través de ***solveLine/3*** (será explicado más adelante) y la guardo en la nueva grilla, de lo contrario copio la columna de la grilla original y voy con la siguiente.

```
paintInitialCols([],[],[]).

paintInitialCols([C|Cs],[L|Ls],[SolvedL|Ns]):-
length(L,Length),
oneSolve(C,Length),
solveLine(C,L,SolvedL),
paintInitialCols(Cs,Ls,Ns),!.

paintInitialCols([_|Cs],[L|Ls],[L|Ns]):-
paintInitialCols(Cs,Ls,Ns).
```

trySolveGrid/4:

Este predicado cuenta con los siguientes parámetros: trySolveGrid(+RowClues,+ColClues,+Grid,-SolvedGrid). Primero busca soluciones para las filas y luego se fija si se satisfacen todas las columnas.

```
trySolveGrid(RowClues,ColClues,Grid,SolvedGrid):-
trySolveRows(RowClues,Grid,SolvedGrid),
transpose(SolvedGrid,TSolvedGrid),
checkCols(ColClues,TSolvedGrid).
```

trySolveRows/3:

Este predicado cuenta con los siguientes parámetros: trySolveRows(+RowClues, +Grid, -SolvedGrid). Trata de resolver cada fila con sus respectivas pistas.

```
trySolveRows([C|_],[R|_],[S]):-
solveLine(C,R,S).

trySolveRows([C|Cs],[R|Rs],[S|Ss]):-
solveLine(C,R,S),
trySolveRows(Cs,Rs,Ss).
```

solveLine/3:

Cuenta con los siguientes parámetros: solveLine(+Clues, +Line,-NewLine). Intenta resolver la línea resolviendo cada una de sus pistas en órden, luego verifica que el resto de la línea esté sin pintar ya que de lo contrario se incumple el control hecho previamente, y lo llena de cruces para

```
%solveLine(Clues,Line,NewLine)
solveLine([],Line,NewLine):-
fillWithCross(Line,NewLine).

solveLine([C|Cs],Line,NewLine):-
trySolveClue(C,0,Line,RestOfLine,L),
solveLine(Cs,RestOfLine,Ls),
append(L,Ls,NewLine).
```

trySolveClue/5:

Este predicado cuenta con los siguientes parámetros: trySolveClue(+Clue, +Gap,+Line,-RestOfLine,-NewLine). Intenta resolver la pista, si no lo consigue y aún tiene espacio suficiente en la línea, lo vuelve a intentar pero incrementando en 1 el espacio inicial.

```
trySolveClue(C,Gap,Line,RestOfLine,NewLine):-
solveClue(C,Gap,Line,RestOfLine,NewLine).

trySolveClue(C,Gap,Line,RestOfLine,NewLine):-
NewGap is Gap+1,
length(Line,Length),
N is NewGap+C,
N =< Length,
trySolveClue(C,NewGap,Line,RestOfLine,NewLine).
```

solveClue/5:

Este predicado tiene los siguientes parámetros: solveClue(+Clue, +Gap,+Line,-RestOfLine,-NewLine). Avanza en la línea hasta que Gap sea 0, luego pinta hasta satisfacer la pista. Aprovecha el pintado inicial, si hay una celda pintada donde estoy haciendo el Gap, entonces no sigo intentando la solución actual. En las celdas sin pintar pone cruces.

```
%solveClue(+C,+Gap,+Line,-RestOfLine,-NewLine)
solveClue(C,0,Line,RestOfLine,NewLine):-
paintClue(C,Line,RestOfLine,NewLine).

solveClue(C,Gap,[H|T],RestOfLine,["X"|L]):-
H \== "#",
Gr is Gap-1,
solveClue(C,Gr,T,RestOfLine,L).
```

paintClue/4:

Este predicado cuenta con los siguientes parámetros: paintClue(+Clue,+Line, -RestOfLine, -NewLine). Pinta la línea hasta que la pista quede satisfecha y aprovecha el pintado inicial para no realizar una solución si se está tratando de pintar en una celda que tiene una “X”, o si hay una celda pintada justo después de donde debería terminar de pintar. En la última celda que va sin pintar pone una cruz.

```
paintClue(0,[],[],[]).

paintClue(0,[H|T],T,["X"]):- H \== "#".

paintClue(Clue,[H|T],RestOfLine,["#"|L]):-
Clue > 0,
H \== "X",
RC is Clue -1,
paintClue(RC,T,RestOfLine,L).
```

Funcionalidades extra:

Botón reiniciar nivel:

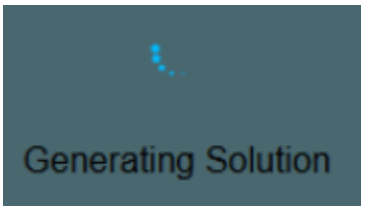
Dejamos un botón en pantalla para que el jugador pueda reiniciar el nivel en el momento que desee.

Vidas del jugador:

Se mostrarán en pantalla en forma de corazones las vidas del jugador, quien inicialmente contará con 3 vidas para resolver el tablero. Si pinta o marca una cruz en una celda que no debe se le restará 1 vida. Al tercer error perderá el nivel y se le dará la opción de jugar otra vez, empezando el nivel desde el principio.

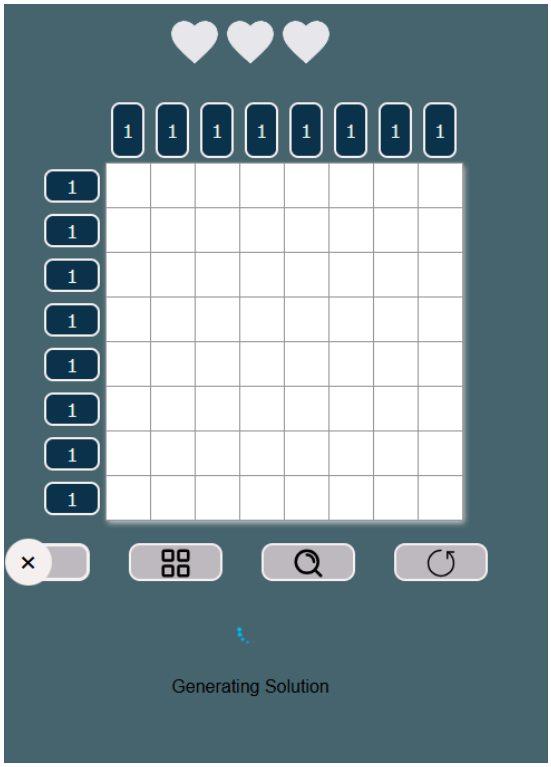
Animación “Generating solution”:

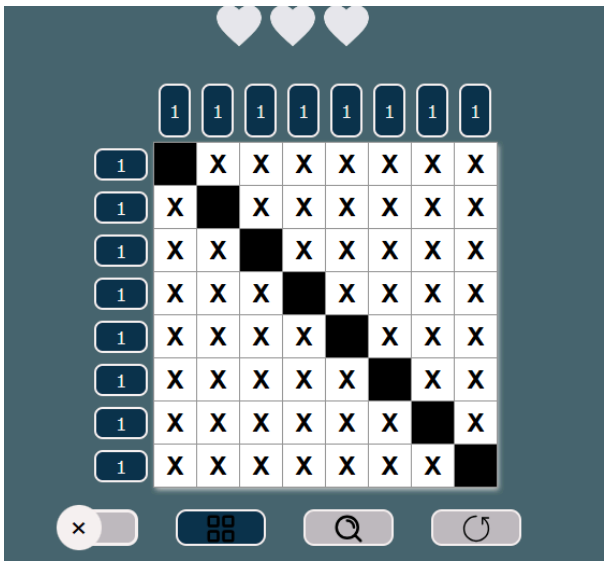
Mientras se está construyendo en en prolog el tablero resuelto, no solo que los botones de ayuda no tendrán efecto y no se podrá pintar el tablero, además aparecerá la siguiente animación en pantalla para saber cuando ya se puede jugar.



Casos de test significativos:

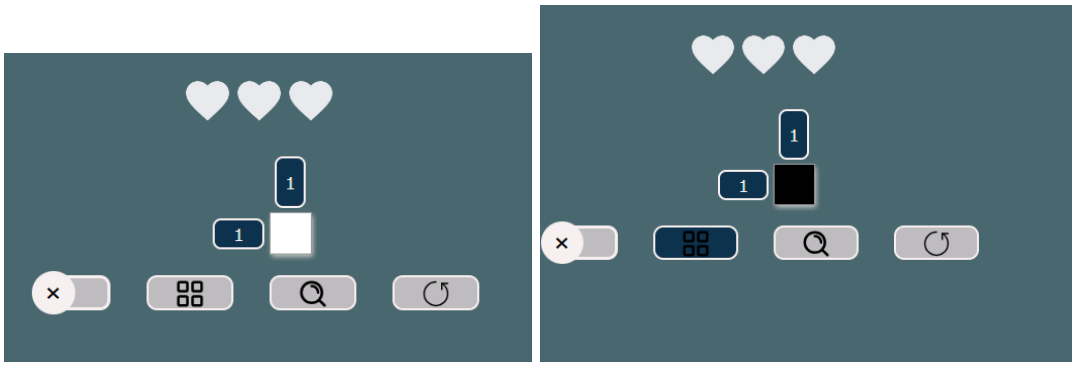
Diagonal en tablero 8x8 (sin pintado inicial):



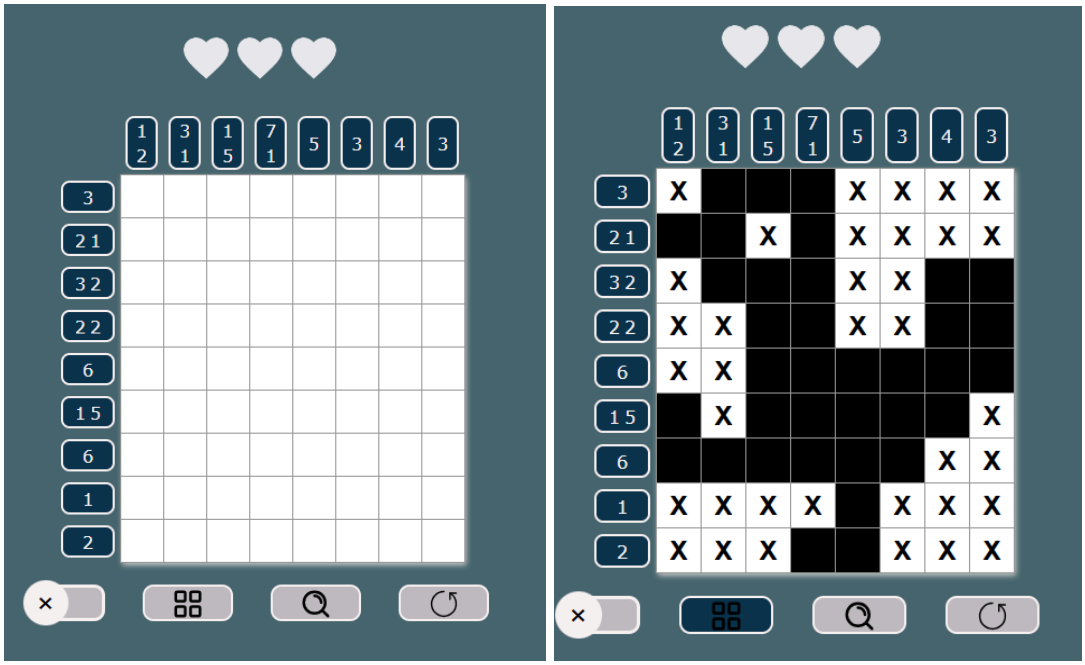


Ya que es un caso costoso, se muestra en pantalla la animación para saber cuando ya se puede jugar.

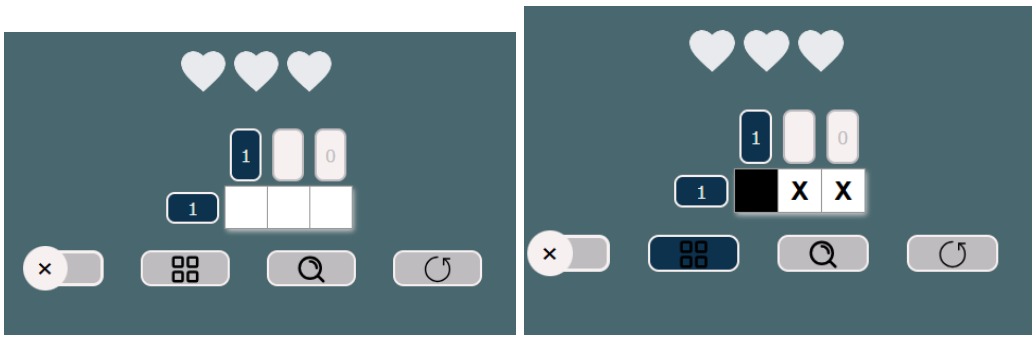
Tablero 1x1:



Matriz rectangular:



Matriz fila o columna con pista 0 y pista vacía:



Tablero grande(15x15):