Implementação do Puzzle *Walls* usando Programação em Lógica com Restrições

José Francisco Cagigal da Silva Gomes e Margarida Xavier Viterbo

FEUP-PLOG, Turma 3MIEIC4, Grupo Walls\_1

Faculdade de Engenharia da Universidade do Porto,

Rua Roberto Frias, sn, 4200-465, Paranhos, Porto, Portugal

{up201305016, [up201403205}@fe.up.pt](mailto:up201403205%7d@fe.up.pt)

<http://www.fe.up.pt>

**Resumo.** Serve o presente artigo como complemento ao segundo trabalho prático da cadeira Programação em Lógica – resolução de um problema de decisão/otimização usando Programação em Lógica com Restrições. Neste caso especifico o problema trata-se de um puzzle 2D, de nome *Walls*, cujo objetivo é colocar linhas verticais ou horizontais em todas as células vazias. As células não vazias contêm números que indicam o número total de segmentos conectados a si (que tocam nas paredes de cada uma das células em causa).

**Keywords:** problemas de decisão, puzzle, PLR, SICStus-Prolog

1. Introdução

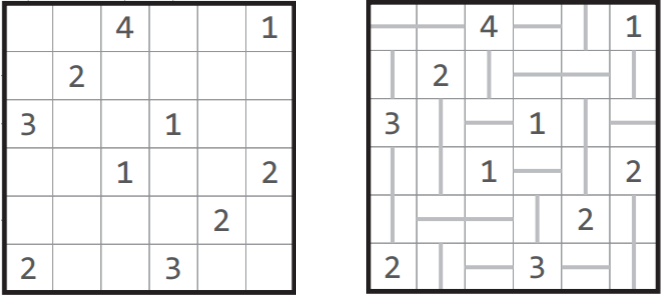
O projeto adotado pelo grupo, *Walls*, para o segundo trabalho prático da cadeira Programação em Lógica, tem como objetivo fundamental dotar os alunos de uma maior capacidade para resolver problemas de decisão, bem como aprofundar o conhecimento a aumentar a aptidão para programação em lógica usando restrições. Através da realização deste projeto os alunos ganham uma destreza para a utilização da bilblioteca ‘clpfd’ presente no SICStus-Prolog (biblioteca-chave para programação em lógica com restrições – PLR).

O desafio em questão, resumido anteriormente, será descrito abaixo. Consta também neste artigo a abordagem utilizada para a resolução do problema, bem como as restrições, função de avaliação e a estratégia de pesquisa. São descritos também a visualização da solução, os resultados e as conclusões.

1. Descrição do Problema

O puzzle *Walls* consiste numa grelha quadrada dividida em células quadradas. Algumas das células contêm números inscritos, outras encontram-se em branco. O tamanho (número de células do tabuleiro) pode variar, tal como os números e a sua disposição, podendo desta forma “jogar-se” com diferentes tabuleiros, criando-se diferentes desafios.

Para resolver o problema é necessário preencher todas as células vazias com traços verticais ou horizontais, de forma que cada célula com número tenha a apontar para si um comprimento total de traços igual ao número, estes traços não têm necessariamente de estar nas células adjacentes à célula numerada, basta que um deles esteja e que os restantes estejam ligados a ele. De forma a esclarecer melhor as regras do puzzle mostra-se abaixo uma figura com um possível puzzle por resolver e a respetiva resolução.



**Fig. 1.** Exemplo de um possível puzzle (esquerda) e sua resolução (direita)

1. Abordagem

Primeiramente foi tentada uma abordagem usando unicamente restrições proposicionais. A lógica implementada era a seguinte: para cada número do tabuleiro, todas as posições de variáveis livres nas 4 direções, até atingir um número ou a borda do tabuleiro, são postas em listas separadas. A soma do número de variáveis de decisão seguidas e iguais (desde o início da lista) em cada lista tinha de ser igual ao número que está a ser analisado.

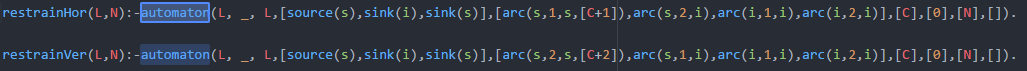
Cedo se conclui que esta abordagem não escalava bem com o tamanho dos tabuleiros. Aproveitando a sugestão do professor Henrique Cardoso, utilizamos a mesma lógica mas substituindo as restrições proposicionais por um autómato.

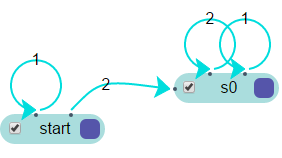
* 1. Variáveis de Decisão

As variáveis de decisão são os elementos do tabuleiro que não são números. O domínio das mesmas pertence ao intervalo [1;2], sendo que 1 significa uma parede horizontal e 2 se traduz numa parede vertical.

* 1. Restrições

Recorremos ao uso do predicado *automaton/9*, onde foi implementado um contador, de forma a sabermos qual é o número de variáveis seguidas. É feito um autómato para cada um dos casos: lista horizontal e lista vertical.





**Fig. 2.** Exemplo de um autómato para as restrições horizontais

Para cada número do tabuleiro são implementadas duas restrições horizontais (lista direita e esquerda do número) e duas restrições verticais (listas acima e abaixo do número).

No final, a soma do contador dos autómatos retornado nos 4 casos tem de ser igual ao número que está a ser analisado (N1):

N1#=VSize1+VSize2+VSize41+VSize31

* 1. Estratégia de Pesquisa

A primeira solução a ser encontrada, será a solução do puzzle que será apresentada. Todas as variáveis de decisão entram no *labeling*:

labeling([], Domain).

* 1. Geração de tabuleiros

Na geração de tabuleiros aleatórios implementou-se um sistema que coloca, numa posição aleatória do tabuleiro, um número também aleatório. Antes de cada inserção verifica-se se o tabuleiro tem solução. Se não tiver, reduz-se o número até não ser possível introduzir um valor naquela posição. A geração só termina após todas as posições tiverem sido testadas.

1. Visualização da Solução

Para visualização dos tabuleiros em modo de texto foram implementados os seguintes predicados:

* ***display\_board***: implementa os predicados *display\_line* e *display\_separator* de forma recursiva, com o objetivo de mostrar o tabuleiro;

display\_board([L1|L2,Size) :-

diplay\_line(L1), write(':'), nl,

display\_separator(Size,1), display\_board(L2,Size).

display\_board([],\_).

* ***display\_line***: imprime o conteúdo de uma linha, escrevendo todos os seus elementos um por um, recursivamente;

display\_line([L1|L2]) :-

write(':'), (var(L1), write(' '), !;

write(L1)), display\_line(L2).

display\_line([]).

* ***display\_separator***: imprime os pontos que fazem separação das células de acordo com o número de células do tabuleiro;

display\_separator(Size, It) :-

It=<Size, !, write('..'), It1 is It+1,

display\_separator(Size,It1).

display\_separator(\_, \_) :- write('.'),nl.

* ***draw\_solution/draw\_solution\_line***: equivalentes aos predicados *display\_board* e *display\_line* mas aplicados ao tabuleiro da solução;

draw\_solution([L1|L2], [L3|L4], Size) :-

draw\_solution\_line(L1, L3), write(':'), nl,

display\_separator(Size, 1), draw\_solution(L2,L4,Size).

draw\_solution([], [], \_).

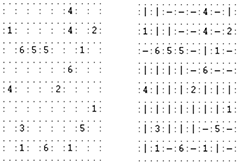
draw\_solution\_line([L1|L2], [L3|L4]) :-

write(':'), (var(L1), (L3==1, write('-'), !;

write('|')), !; write(L1)), draw\_solution\_line(L2,L4).

draw\_solution\_line([], []).

Estes predicados de visualização imprimem o seguinte:



**Fig. 3.** Visualização em modo texto dos tabuleiros inicial (esquerda) e final (direita) do puzzle

1. Resultados

A tabela abaixo diz respeito aos resultados obtidos ao encontrar uma solução para um tabuleiro previamente definido (não envolve geração de tabuleiro). Podemos observar que quanto maior o tabuleiro, maior poderá ser o número de restrições, e consequentemente maior será o tempo de resolução.

|  |  |  |  |
| --- | --- | --- | --- |
| **Tamanho puzzle** | **Nr. restrições** | **Tempo solução (ms)** | **Backtracks** |
| 6x6 | 174 | 8 | 5 |
| 10x10 | 480 | 27 | 35 |
| 15x15 | 1256 | 75 | 313 |
| 20x20 | 2320 | 129 | 991 |

**Tabela 1.** Tabela com os resultados relativos à solução de um tabuleiro pré-feito.

A implementação de uma solução incluindo a geração de um tabuleiro aleatório acabou por ser pouco ineficiente para tabuleiros maiores. Como se pode verificar após a análise da tabela 2., quanto maior o tabuleiro maior o tempo de geração.

|  |  |  |  |
| --- | --- | --- | --- |
| **Tamanho puzzle** | **Nr. restrições** | **Tempo solução (s)** | **Backtracks** |
| 6x6 | 7954 | 0.693 | 139 |
| 10x10 | 168286 | 31.792 | 498493 |
| 15x15 | 1018633 | 95.527 | 2257303 |
| 20x20 | 4782138 | 411.342 | 3200804 |

**Tabela 2.** Tabela com os resultados relativos à solução de um tabuleiro gerado automaticamente.

1. Conclusões

Com a realização deste trabalho foi possível perceber a utilidade prática da programação em lógica com restrições e de que forma esta pode facilitar imenso e resolução de problemas de decisão. Foi igualmente conseguida uma ambientação e destreza claras com este tipo de programação e bibliotecas associadas.

Embora no início se tenha revelado um pouco complicado alterar o raciocino lógico do trabalho anterior e procurar soluções usando restrições, ao começar a ser explorada esta nova forma de programação em *prolog*, verificou-se a enorme diferença entre o resultado da implementação do puzzle com e sem restrições, tendo-se facilmente percebido a importância da PLR.

O problema proposto foi resolvido com sucesso e generalizado para todos os casos. O que começou por ser uma resolução bastante demorada foi otimizado e é agora uma solução eficiente para o puzzle em questão.

1. Bibliografia

[1] Regras do puzzle *Walls*,

<http://logicmastersindia.com/lmitests/dl.asp?attachmentid=523>

[2] SICStus Prolog, <https://sicstus.sics.se/>

[3] SWI-Prolog, <http://www.swi-prolog.org/>

[4] Elaboração do artigo,

<ftp://ftp.springer.de/pub/tex/latex/llncs/latex2e/instruct/authors/authors.pdf>

Anexos

Ficheiro *main.pl*:

:- include('generator.pl').

:- include('aula.pl').

%modo de puzzle

choosePuzzleStyle(X):-write('1-Choose board'),nl,write('2-RandomBoard'),nl,prcss\_ans(1,2,X).

%predicado de começo

start:-choosePuzzleStyle(X),playThepuzzle(X).

%pergunta qual o tabuleiro que quer ver resolvido e pede para se resolver

playThepuzzle(1):-askId(ID),board1(ID,L),init(ID,L).

%pede id do tabuleiro que quer ver resolvido ao utilizador

askId(ID):-write('Choose Id'),nl,prcss\_ans(2,50,ID).

%gera um tabuleiro com um tamanho arbitrário e pede para resolver

playThepuzzle(2):-askSize(Size),generate(Size,L),init(Size,L).

%pergunta o tamanho ao utilizador

askSize(Size):-write('Choose Size of Board'),nl,prcss\_ans(2,50,Size).

%processa resposta do utilizador

prcss\_ans(Min,Max,Ans):-read\_line(X),lineToNumber(X,Ans),Ans>=Min,Ans=<Max.

%auxiliar para processar resposta do utilizador

lineToNumber([X1|X2],Ans):-lineToNumber(X2,Ans1),(Ans1 \= -1,Ans is (X1-48)\*10+Ans1;Ans is X1-48).

lineToNumber([],-1).

Ficheiro *generator.pl*:

:- use\_module(library(random)).

:- use\_module(library(timeout)).

%cria uma matriz (SizexSize) vazia e devolve na Variavel List

createEmptyMatrix(Size,List):-createEmptyMatrixAux(Size,List,Size).

%predicado auxiliar para criar uma matrix vazia

createEmptyMatrixAux(Size,[NewList|List2],Counter):-Counter>0,Counter1 is Counter-1,length(NewList,Size),createEmptyMatrixAux(Size,List2,Counter1).

createEmptyMatrixAux(\_,[],0).

:-dynamic matrixPos/1.

matrixPos([]).

%põe todas as posições do tabuleiro num facto dinâmico matrixPos

createMatrixPositions([L1|L2],X):-createMatrixPositionsLine(L1,X,1),X1 is X+1,createMatrixPositions(L2,X1).

createMatrixPositions([],\_).

%auxiliar para preencher matrixPos

createMatrixPositionsLine([\_|L2],X,Y):-matrixPos(List),append(List,[[X,Y]],NewList),retract(matrixPos(\_)),asserta(matrixPos(NewList)),Y1 is Y+1,createMatrixPositionsLine(L2,X,Y1).

createMatrixPositionsLine([],\_,\_).

%devolve um puzzle aleatório em L, com o tamanho SizexSize

generate(Size,L):-createEmptyMatrix(Size,L),createMatrixPositions(L,1),createBoard(L,Size).

%vai a uma posição aleatória do tabuleiro para colocar lá um número

createBoard(L,Size):-matrixPos(List),length(List,Length),Length>0,Range is Length+1,MaxNumber is round(3\*Size/4)+1,random(1,Range,Random),placeElement(L,Random,MaxNumber,Size),!,createBoard(L,Size).

createBoard(\_,\_).

%pede pra colocar um número random na posição escolhida anteriormente

placeElement(L,Element,MaxNumber,Size):-random(1,MaxNumber,Random),matrixPos(List),nth1(Element,List,RandomPos),!,tryToPlace(L,RandomPos,Random,Size).

%tenta colocar o número escolhido na posiçao escolhida e verifica se é possivel, se não fôr, reduz o número até ser impossível colocar na posição um número

tryToPlace(L,[X,Y],Random,Size):-Random>0,copy(L,LL),getElement(LL, X, Y, Value),Value=Random,(time\_out(main(LL,Size), 100,TimeOut),TimeOut=success,!,getElement(L, X, Y, ValueReal),ValueReal=Random,matrixPos(ListPositions),delete(ListPositions,[X,Y],NewList),retract(matrixPos(\_)),asserta(matrixPos(NewList))

;!,Random1 is Random -1,tryToPlace(L,[X,Y],Random1,Size)).

tryToPlace(\_,[X,Y],0,\_):-!,matrixPos(ListPositions),delete(ListPositions,[X,Y],NewList),retract(matrixPos(\_)),asserta(matrixPos(NewList)).

%recebe uma matriz e devolve a mesma matrix com variaveis diferentes (números mantêm-se iguais)

copy([L1|L2],[NewList1|NewList2]):-copyLine(L1,NewList1),copy(L2,NewList2).

copy([],[]).

%predicado auxiliar para copiar a linha de uma matriz

copyLine([L1|L2],[NewList1|NewList2]):-var(L1),NewList1=Var,copyLine(L2,NewList2).

copyLine([L1|L2],[NewList1|NewList2]):-NewList1=L1,copyLine(L2,NewList2).

copyLine([],[]).

Ficheiro *aula.pl*:

:- use\_module(library(clpfd)).

:- use\_module(library(lists)).

:-dynamic board1/2.

board1(2,[[1,1],[B,1]]).

board1(6,[[A,B,4,C,D,1],[E,2,F,G,H,I],[3,J,K,1,L,M],[N,O,1,P,Q,2],[R,S,T,U,2,V],[2,X,Y,3,W,Z]]).

board1(10,[[A,A1,2,A2,A3,A4,A5,A6,A7,6],[B,B1,B2,B3,6,B4,B5,B6,B7,B8],

[C,6,C1,C2,3,C4,C5,8,7,6],[D,D1,1,D2,D3,D4,D5,D6,D7,D8],[1,E1,E2,E3,E4,7,E5,E6,E7,E8],

[1,F1,1,F2,1,F4,F5,F6,F7,F8],[G,G1,4,G2,G3,G4,3,G6,G7,G8],[1,4,H1,H2,H3,H4,4,H6,H7,H8],

[I,I1,I2,I3,I4,6,I5,I6,I7,6],[1,J1,6,J2,J3,J4,J5,J6,J7,J8]]).

board1(15,[[A1,4,A3,A4,A5,7,A7,A8,A9,A10,4,3,A13,A14,A15],[B1,B2,B3,B4,B5,B6,B7,B8,B9,11,B11,B12,B13,B14,1],

[7,C2,C3,C4,C5,C6,C7,1,C9,C10,C11,3,C13,4,C15],[D1,D2,D3,D4,D5,8,6,D8,D9,D10,D11,D12,5,D14,D15],

[E1,E2,1,E4,2,E6,4,E8,E9,E10,E11,E12,E13,6,E15],[F1,3,F3,F4,F5,F6,F7,F8,6,F10,F11,3,F13,F14,F15],

[2,H2,H3,4,H5,H6,H7,H8,H9,H10,H11,2,H13,H14,H15],[2,G2,2,G4,1,G6,3,G8,G9,3,G11,G12,G13,G14,G15],

[I1,I2,4,I4,I5,I6,I7,I8,3,I10,I11,I12,6,I14,6],[8,J2,J3,J4,J5,5,J7,J8,J9,J10,5,J12,4,J14,J15],

[K1,K2,K3,10,K5,K6,K7,7,K9,K10,K11,K12,K13,K14,K15],[L1,L2,L3,L4,L5,L6,L7,L8,7,L10,L11,L12,L13,L14,3],

[M1,M2,1,M4,3,M6,6,M8,M9,M10,M11,M12,9,M14,M15],[N1,4,N3,N4,N5,1,N7,8,N9,N10,N11,N12,N13,N14,N15],

[O1,O2,O3,O4,O5,O6,O7,O8,O9,O10,O11,9,O13,O14,O15]]).

board1(20,[[1,A2,2,A4,A5,A6,1,A8,A9,9,A11,A12,A13,A14,A15,A16,A17,A18,9,A20],

[B1,B2,B3,B4,B5,5,B7,B8,1,B10,B11,B12,6,B14,B15,B16,B17,B18,B19,B20],

[1,C2,C3,C4,C5,C6,C7,13,C9,C10,C11,C12,C13,C14,C15,C16,C17,5,C19,3],

[D1,12,D3,D4,D5,D6,D7,9,D9,D10,D11,D12,D13,D14,13,D16,D17,D18,1,D20],

[2,E2,E3,E4,E5,E6,E7,E8,E9,11,E11,E12,E13,E14,E15,E16,1,E18,1,E20],

[F1,F2,12,F4,F5,F6,F7,F8,F9,F10,F11,F12,F13,F14,F15,F16,6,5,F19,F20],

[H1,H2,H3,5,H5,H6,H7,H8,H9,6,2,H12,2,H14,H15,H16,H17,H18,H19,H20],

[2,10,G3,G4,G5,G6,G7,G8,G9,G10,G11,G12,2,G14,G15,3,G17,4,3,5],

[10,I2,I3,I4,I5,I6,I7,I8,I9,I10,I11,I12,4,I14,I15,I16,I17,I18,I19,I20],

[J1,J2,2,2,J5,J6,2,3,J9,J10,J11,5,J13,1,J15,J16,J17,J18,J19,J20],

[10,K2,K3,K4,K5,K6,K7,K8,K9,K10,K11,K12,K13,K14,K15,2,K17,K18,4,K20],

[L1,L2,L3,L4,5,2,L7,L8,3,L10,2,L12,L13,2,L15,L16,L17,12,L19,L20],

[M1,M2,M3,3,M5,M6,M7,M8,M9,7,M11,M12,M13,M14,M15,M16,M17,M18,M19,M20],

[N1,N2,N3,N4,N5,N6,14,N8,N9,N10,N11,N12,N13,N14,N15,3,N17,N18,N19,7],

[3,O2,O3,O4,4,O6,O7,O8,7,O10,O11,O12,O13,O14,O15,O16,O17,O18,2,O20],

[P1,P2,P3,P4,P5,P6,15,P8,P9,P10,P11,P12,P13,P14,P15,2,P17,P18,P19,5],

[Q1,Q2,Q3,4,Q5,Q6,Q7,4,Q9,Q10,Q11,Q12,6,Q14,3,1,Q17,Q18,Q19,Q20],

[R1,R2,R3,R4,R5,R6,R7,R8,R9,R10,R11,10,R13,R14,R15,1,R17,R18,4,R20],

[6,S2,S3,S4,S5,S6,S7,S8,S9,S10,S11,S12,S13,S14,11,S16,1,S18,S19,S20],

[T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,13,T14,T15,4,T17,T18,T19,T20]]).

%board1(15,[[A1,A2,A3,A4,A5,A6,A7,A8,A9,A10,A11,A12,A13,A14],[B1,B2,B3,B4,B5,B6,B7,B8,B9,B10,B11,B12,B13,B14],

%[C1,C2,C3,C4,C5,C6,C7,C8,C9,C10,C11,C12,C13,C14],[D1,D2,D3,D4,D5,D6,D7,D8,D9,D10,D11,D12,D13,D14],

%[E1,E2,E3,E4,E5,E6,E7,E8,E9,E10,E11,E12,E13,E14],[F1,F2,F3,F4,F5,F6,F7,F8,F9,F10,F11,F12,F13,F14],

%[H1,H2,H3,H4,H5,H6,H7,H8,H9,H10,H11,H12,H13,H14],[G1,G2,G3,G4,G5,G6,G7,G8,G9,G10,G11,G12,G13,G14],

%[I1,I2,I3,I4,I5,I6,I7,I8,I9,I10,I11,I12,I13,I14],[J1,J2,J3,J4,J5,J6,J7,J8,J9,J10,J11,J12,J13,J14],

%[K1,K2,K3,K4,K5,K6,K7,K8,K9,K10,K11,K12,K13,K14],[L1,L2,L3,L4,L5,L6,L7,L8,L9,L10,L11,L12,L13,L14],

%[M1,M2,M3,M4,M5,M6,M7,M8,M9,M10,M11,M12,M13,M14],[N1,N2,N3,N4,N5,N6,N7,N8,N9,N10,N11,N12,N13,N14],

%[O1,O2,O3,O4,O5,O6,O7,O8,O9,O10,O11,O12,O13,O14]]).

board1(15,[[14,A2,A3,A4,A5,A6,A7,A8,A9,A10,A11,A12,A13,A14,A15],[14,B2,B3,B4,B5,B6,B7,B8,B9,B10,B11,B12,B13,B14,B15],

[14,C2,C3,C4,C5,C6,C7,C8,C9,C10,C11,C12,C13,C14,C15],[14,D2,D3,D4,D5,D6,D7,D8,D9,D10,D11,D12,D13,D14,D15],

[14,E2,E3,E4,E5,E6,E7,E8,E9,E10,E11,E12,E13,E14,E15],[14,F2,F3,F4,F5,F6,F7,F8,F9,F10,F11,F12,F13,F14,F15],

[14,H2,H3,H4,H5,H6,H7,H8,H9,H10,H11,H12,H13,H14,H15],[14,G2,G3,G4,G5,G6,G7,G8,G9,G10,G11,G12,G13,G14,G15],

[14,I2,I3,I4,I5,I6,I7,I8,I9,I10,I11,I12,I13,I14,I15],[14,J2,J3,J4,J5,J6,J7,J8,J9,J10,J11,J12,J13,J14,J15],

[14,K2,K3,K4,K5,K6,K7,K8,K9,K10,K11,K12,K13,K14,K15],[14,L2,L3,L4,L5,L6,L7,L8,L9,L10,L11,L12,L13,L14,L15],

[14,M2,M3,M4,M5,M6,M7,M8,M9,M10,M11,M12,M13,M14,M15],[14,N2,N3,N4,N5,N6,N7,N8,N9,N10,N11,N12,N13,N14,N15],

[14,O2,O3,O4,O5,O6,O7,O8,O9,O10,O11,O12,O13,O14,O15]]).

board1(3,[[A,B,C],[D,E,F],[G,3,H],[I,J,K]]).4,O15]]).

board1(3,[[A,B,C],[D,E,F],[G,3,H],[I,J,K]]).

%aplicar restrições horizontais a uma lista

restrainHor(L,N):-automaton(L, \_, L,[source(s),sink(i),sink(s)],[arc(s,1,s,[C+1]),arc(s,2,i),arc(i,1,i),arc(i,2,i)],[C],[0],[N],[]).

%aplicar restrições verticais a uma lista

restrainVer(L,N):-automaton(L, \_, L,[source(s),sink(i),sink(s)],[arc(s,2,s,[C+2]),arc(s,1,i),arc(i,1,i),arc(i,2,i)],[C],[0],[N],[]).

%mostrar o puzzle no ecrã

display\_board([L1|L2],Size):-display\_line(L1),write(':'),nl,display\_separator(Size,1),display\_board(L2,Size).

display\_board([],\_).

%auxiliar para mostrar o puzzle no ecrã

display\_line([L1|L2]):-write(':'),(var(L1),write(' '),!;write(L1)),display\_line(L2).

display\_line([]).

%linha separadora (efeito visual)

display\_separator(Size,It):-It=<Size,!,write('..'),It1 is It+1,display\_separator(Size,It1).

display\_separator(\_,\_):-write('.'),nl.

%inicia os devidos campos e pede para se resolver o puzzle

init(Size,L):-retract(board1(\_,\_)),asserta(board1(Size,L)),display\_separator(Size,1),display\_board(L,Size),nl,nl,main(L,Size),board1(\_,OldPuzzle),display\_separator(Size,1),draw\_solution(OldPuzzle,L,Size).

%resolve o puzzle

main(Matrix,Size):-findNumberPaths(Matrix,1,Size,Paths,NumberArray),calculateDomain(Matrix,Domain),puzzle(Paths,NumberArray,Domain).

%predicado para desenhar a solução do puzzle

draw\_solution([L1|L2],[L3|L4],Size):-draw\_solution\_line(L1,L3),write(':'),nl,display\_separator(Size,1),draw\_solution(L2,L4,Size).

draw\_solution([],[],\_).

%auxiliar para desenhar a solução do puzzle

draw\_solution\_line([L1|L2],[L3|L4]):-write(':'),(var(L1),(L3==1,write('-'),!;write('|')),!;write(L1)),draw\_solution\_line(L2,L4).

draw\_solution\_line([],[]).

%declarar dominio, restrições e o labeling

puzzle(Paths,NumberArray,Domain):-domain(Domain,1,2),restrain(Paths,NumberArray),labeling([],Domain).

%todas as restriçoes relativas ao puzzle estão aqui e são chamadas por este predicado (apenas faz restrições)

restrain([Path1,Path2,Path3,Path4|Paths],[N1|NR]):-

length(Path1,TotalSize1),domain([VSize1],0,TotalSize1),restrainHor(Path1,VSize1),

length(Path2,TotalSize2),domain([VSize2],0,TotalSize2),restrainHor(Path2,VSize2),

length(Path3,TotalSize3),TotalSize31 is 2\*TotalSize3,domain([VSize3],0,TotalSize31),restrainVer(Path3,VSize3),

length(Path4,TotalSize4),TotalSize41 is 2\*TotalSize4,domain([VSize4],0,TotalSize41),restrainVer(Path4,VSize4),

VSize31 #= VSize3/2, VSize41 #=VSize4/2,

N1#=VSize1+VSize2+VSize41+VSize31,restrain(Paths,NR).

restrain([],[]).

%predicado que recebe um tabuleiro e retorna uma lista com as variáveis não declaradas sem repetições (= tabuleiro sem os números)

calculateDomain([L1|L2],R):-calculateDomainAux(L1,R1),calculateDomain(L2,R2),append(R1,R2,R).

calculateDomain([],[]).

%predicado auxiliar de caulculateDomain

calculateDomainAux([L1|L2],R):-number(L1),!,calculateDomainAux(L2,R).

calculateDomainAux([L1|L2],[L1|R2]):-calculateDomainAux(L2,R2).

calculateDomainAux([],[]).

%retorna uma lista de listas com as variáveis não declaradas

findNumberPaths(Matrix,X,Size,LR,NumberPath):-length(Matrix,Tamanho),Tamanho>=X,!,findNumberPathsAux(Matrix,[X,1],Size,L1,Number1),X1 is X+1,findNumberPaths(Matrix,X1,Size,L2,Number2),

append(L1,L2,LR),append(Number1,Number2,NumberPath).

findNumberPaths(\_,\_,\_,[],[]).

%para um dado número do puzzle encontra todas as variáveis que estão à sua esquerda, direita,cima e baixo e coloca numa lista de listas, sendo cada caminho uma lista

findNumberPathsAux(Matrix,[X,Y],Size,[L1,L2,L3,L4|LR],[Value|Nr]):-Y=<Size,getElement(Matrix, X, Y, Value), number(Value),!,

findHorizontalPath(Matrix,[X,Y],Size,L1,L2),length(Matrix,SizeVer),findVerticalPath(Matrix,[X,Y],SizeVer,L3,L4),

Y1 is Y+1,findNumberPathsAux(Matrix,[X,Y1],Size,LR,Nr).

findNumberPathsAux(Matrix,[X,Y],Size,LR,Nr):-Y=<Size,Y1 is Y+1,!,findNumberPathsAux(Matrix,[X,Y1],Size,LR,Nr).

findNumberPathsAux(\_,[\_,\_],\_,[],[]).

%pede os caminhos de um número para a esquerda e para a direita e devolve-os

findHorizontalPath(Matrix,[X,Y],Size,L1,L2):-findRightPath(Matrix,[X,Y],Size,L1),findLeftPath(Matrix,[X,Y],Size,L2).

%procura o caminho para a direita e devolve

findRightPath(Matrix,[X,Y],Size,[Value|L2]):-Y1 is Y+1, Y1=<Size, getElement(Matrix, X, Y1, Value), \+ number(Value),!,findRightPath(Matrix, [X,Y1],Size,L2).

findRightPath(\_,[\_,\_],\_,[]).

%procura o caimnho para a esquerda e devolve

findLeftPath(Matrix,[X,Y],\_,[Value|L2]):-Y1 is Y-1, Y1>0, getElement(Matrix, X, Y1, Value), \+ number(Value),!,findLeftPath(Matrix, [X,Y1],\_,L2).

findLeftPath(\_,[\_,\_],\_,[]).

%pede os caminhos de um número para cim e para baixo e devolve-os

findVerticalPath(Matrix,[X,Y],Size,L1,L2):-findUpPath(Matrix,[X,Y],Size,L1),findLowPath(Matrix,[X,Y],Size,L2).

%procura o caminho para baixo e devolve

findLowPath(Matrix,[X,Y],Size,[Value|L2]):-X1 is X+1, X1=<Size, getElement(Matrix, X1, Y, Value), \+ number(Value),!,findLowPath(Matrix, [X1,Y],Size,L2).

findLowPath(\_,[\_,\_],\_,[]).

%procura o caminho para cima e devolve

findUpPath(Matrix,[X,Y],\_,[Value|L2]):-X1 is X-1, X1>0, getElement(Matrix, X1, Y, Value), \+ number(Value),!,findUpPath(Matrix, [X1,Y],\_,L2).

findUpPath(\_,[\_,\_],\_,[]).

% Retorna elemento de uma posicao X,Y

getElement(Matrix, Row, Col, Value):-

nth1(Row, Matrix, MatrixRow),

nth1(Col, MatrixRow, Value).