Resolução de Problema de Decisão/Otimização usando Programação em Lógica com Restrições

Bosnian Snake

Beatriz Henriques[1] e Beatriz Velho[1]

[1] FEUP-PLOG, Turma 3MIEIC02, Grupo Bosnian Snake_1

Faculdade de Engenharia da Universidade do Porto
Rua Roberto Frias, sn, 4200-465 Porto, Portugal

Resumo. Este artigo complementa o segundo trabalho prático da Unidade Curricular de Programação em Lógica do Mestrado Integrado em Engenharia Informática e de Computação. O objetivo deste trabalho é a construção de um programa em *PROLOG* com restrições para a resolução de um problema de otimização ou decisão combinatória. O problema de decisão escolhido é baseado num puzzle lógico denominado *Bosnian Snake*. Este jogo tem como principal objetivo desenhar a cobra através de pistas, número de linhas e colunas ocupados por esta. Neste artigo mostramos que foi possível a resolução deste problema de forma eficiente, através da manipulação de predicados disponibilizados pelo SICStus Prolog.

Palavras-chave: prolog, restrições, feup, plog, bosnian snake, tabuleiro

1 Introdução

Este projeto foi desenvolvido no âmbito da unidade curricular de Programação em Lógica. Consiste na resolução de um problema de otimização ou decisão combinatória através do uso da biblioteca *clpfd* presente no SICStus Prolog. O problema selecionado pelo grupo foi um puzzle chamado Bosnian Snake.

Este puzzle consiste num jogo de lógica e estratégia, jogado num tabuleiro 8x8. O objetivo do jogo é descobrir a cobra escondida no tabuleiro. Sabendo a localização da cabeça e da cauda da cobra e através de pistas localizadas no tabuleiro, os quadrados ocupados pela cobra, e fora do tabuleiro, o número de quadrados linhas/colunas onde se encontra a cobra. A cobra não pode tocar-se mesmo em diagonais e o caminho tem de ser contínuo.

Este artigo explica o problema em questão detalhadamente e demonstra a resolução deste problema em detalhe.

2 Descrição do Problema

O puzzle *Bosnian Snake* é jogado num tabuleiro quadrado com dimensão de 8x8. Neste são colocadas pistas, a cabeça e a cauda da cobra. As pistas são representadas por números de 1 a 8 e colocadas nas células e à volta do tabuleiro. As pistas colocadas nas células representam o número de células que a cobra irá ocupar em volta dessa mesma célula. E as que se encontram à volta representam o número de células que a cobra pode ocupar na linha/coluna. O corpo da cobra também tem de obedecer algumas regras. Como por exemplo, esta nunca se pode tocar, nem mesmo em diagonal ou a solução deve garantir que o corpo da cobra é um caminho contínuo.

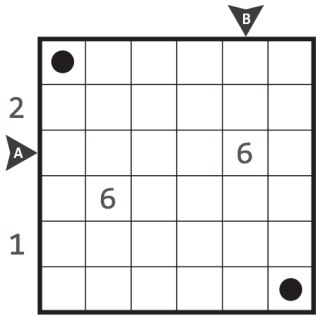


Figura 1 Um possível início de jogo

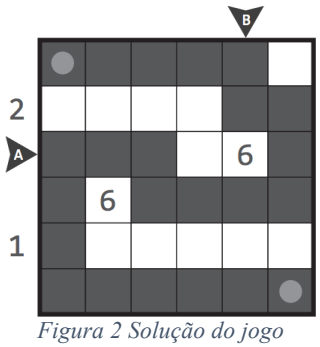


Figura 2 Solução do jogo

3 Abordagem

O primeiro passo foi fazer uma avaliação ponderada de como abordar o puzzle *Bosnian Snake* como um problema de restrições. De seguida, foi necessário definir as variáveis de decisão para aplicar no predicado de *labeling*, a melhor restrição possível e modo mais intuitivo para o utilizador.

O jogo implementado permite tabuleiros de diferentes tamanhos. Os tamanhos predefinidos são 6x6, 9x9, 12x12, no entanto, no jogo *random* é possível escolher o tamanho.

Para tabuleiro é representado através de uma lista de listas, em que cada posição possui um número de -1 a 7 em que:

- -1 representa uma célula ocupada pela cobra;
- 0 representa caminho livre;
- 1 a 7 representa as pistas interiores.

```
[ [-1,0,0,0,0,0], [0,0,0,0,0,0], [0,0,0,0,6,0],
[0,6,0,0,0,0], [0,0,0,0,0,0], [0,0,0,0,0,-1] ]
```

Excerto de código 1 Exemplo lista de listas do tabuleiro 6x6

3.1 Variáveis de Decisão

Para a resolução do jogo Bosnian Snake é utilizada uma variável de decisão (ou variável de domínio) que representa o tabuleiro final, ou seja, o tabuleiro com a solução. Esta variável é uma lista de listas. Cada variável de domínio dentro desta representa uma célula do tabuleiro e o seu domínio é -1 ou 0. O predicado a seguir apresentado é um excerto da definição do domínio das variáveis de decisão.

```
inicializarLinha([_|T],[H1|T1]):-
        H1 in -1..0,
        inicializarLinha(T,T1).
```

Excerto de código 2 Definição do domínio das variáveis de decisão

3.2 Restrições

A resolução do problema resume-se em cinco restrições:

1. As posições inicial e final da cobra só podem ter uma célula vizinha ocupada pela cobra, e esta tem que estar numa das posições ortogonais da célula;
2. O número de células ocupadas pela cobra numa determinada linha ou coluna tem que ser igual à pista exterior;
3. O número de células ocupadas pela cobra em volta de uma casa com um número de 1 a 7 tem que exatamente igual a esse número;
4. Uma célula ocupada pela cobra (exceto a cabeça e cauda) tem que ter exatamente duas células ortogonais vizinhas ocupadas pela cobra;
5. A cobra não pode tocar em si mesma, o que implica também não haver diagonais ocupadas pela cobra.

De seguida são explicadas as restrições acima referidas, sendo apresentado excertos de código utilizado para a sua implementação.

As posições inicial e final da cobra só podem ter uma célula vizinha ocupada pela cobra, e esta tem que estar numa das posições ortogonais da célula.

Independentemente do tamanho do tabuleiro, a cabeça e a cauda da cobra apenas podem ter uma célula vizinha ocupada pela cobra, em que a posição tem que ser ortogonal. O predicado seguinte é responsável por esta restrição.

```
validarInicioFimCobra(TAB,NLINHA,NCOLUNA):-
        getElementosVizinhos(TAB,[[-1,0],[0,1],[1,0],[0,-1]], NLINHA,
NCOLUNA, LISTAVIZINHOS),
        getCasaTabuleiro(TAB,NLINHA,NCOLUNA,CASA),
        count(-1,LISTAVIZINHOS,#=,COUNT),
        CASA #= -1 #=> COUNT #= 1.
```

Excerto de código 3

O número de células ocupadas pela cobra numa determinada linha ou coluna tem que ser igual à pista exterior.

Neste predicado é recebido no segundo parâmetro uma lista com as posições das linhas que têm uma pista exterior ao tabuleiro e o respetivo número de células que a cobra pode ocupar nessa linha. O mesmo predicado é feito para as colunas, sendo feito antes `transpose` do tabuleiro.

```
validarCasasCobraLinha(_,[]).
validarCasasCobraLinha(TAB,[LINHA-COBRALINHA|T]):-
        nth1(LINHA,TAB,LINHA_TAB),
        sum(LINHA_TAB,#=,COBRALINHA1),
        COBRALINHA1 #= (COBRALINHA * -1),
        validarCasasCobraColuna(TAB,T).
```

Excerto de código 4

O número de células ocupadas pela cobra em volta de uma casa com um número de 1 a 7 tem que exatamente igual a esse número.

O primeiro parâmetro no seguinte predicado é o tabuleiro inicial, e por cada célula que tenha um número *H* maior que zero, é definida a restrição de que no tabuleiro solução, o número de células ocupadas pela cobra em volta da célula com a mesma posição é igual a *H*.

```
validarCasasCobraAroundLinha([H|T],TAB1,NLINHA,NCOLUNA):-
        H > 0,
        getElementosVizinhos(TAB1,[[-1,-1],[-1,0],[-1,1],[0,-1],[0,1],[1,-
1],[1,0],[1,1]],NLINHA,NCOLUNA,LISTAVIZINHOS),
        getCasaTabuleiro(TAB1,NLINHA,NCOLUNA,CASA),
        count(-1,LISTAVIZINHOS,#=,COUNT),
        CASA #= 0 #=> COUNT #= H,
        NCOLUNA1 is NCOLUNA + 1,
        validarCasasCobraAroundLinha(T,TAB1,NLINHA,NCOLUNA1).
```

Excerto de código 5

Uma célula ocupada pela cobra (exceto a cabeça e cauda) tem que ter exatamente duas células ortogonais vizinhas ocupadas pela cobra.

Neste predicado é definida a restrição de que cada célula ocupada pela cobra tem que ter exatamente duas casas ortogonais vizinhas ocupadas também pela cobra.

```
validarDoisVizinhosCobra([_|T],TAB,NLINHA,NCOLUNA,[LI/CI,LF/CF]):-
        getElementosVizinhos(TAB,[[-1,0],[0,1],[1,0],[0,-
1]],NLINHA,NCOLUNA,LISTAVIZINHOS),
        getCasaTabuleiro(TAB,NLINHA,NCOLUNA,CASA),
        count(-1,LISTAVIZINHOS,#=,COUNT),
        CASA #= -1 #=> COUNT #= 2,
        NCOLUNA1 is NCOLUNA + 1,
        validarDoisVizinhosCobra(T,TAB,NLINHA,NCOLUNA1,[LI/CI,LF/CF]).
```

Excerto de código 6

A cobra não pode tocar em si mesma, o que implica também não haver diagonais ocupadas pela cobra.

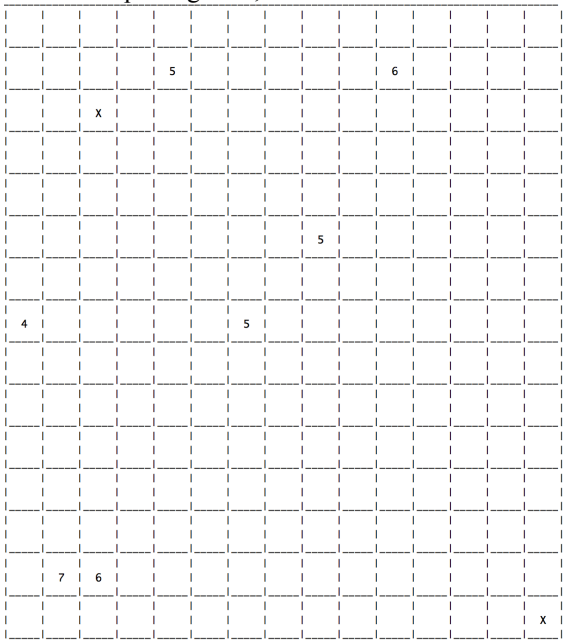
No excerto de código seguinte é definida a restrição de que a cobra não se pode tocar. Para isso, por cada célula com posição *NLINHA/NCOLUNA* do tabuleiro são verificadas as suas diagonais e caso a célula *CASA* e a diagonal *CASA1* em análise sejam parte da cobra, então existe uma única célula vizinha a *CASA* e *CASA1* que é necessariamente também parte da cobra.

```
validarDiagonalNaoDiagonalRestricao(TAB,NLINHA,NCOLUNA,NLINHA1,NCOLUNA1):-
        getCasaTabuleiro(TAB,NLINHA,NCOLUNA,CASA),
        getCasaTabuleiro(TAB,NLINHA1,NCOLUNA1,CASA1),
        getCasaTabuleiro(TAB,NLINHA1,NCOLUNA,CASAVIZINHA1),
        getCasaTabuleiro(TAB,NLINHA,NCOLUNA1,CASAVIZINHA2),
        (CASA #= -1 #/\ CASA1 #= -1) #=>
        (CASAVIZINHA1 #= -1 #/\ CASAVIZINHA2 #= 0)
        #\/
        (CASAVIZINHA1 #= 0 #/\ CASAVIZINHA2 #= -1).
```

Excerto de código 7

3.3 Jogo Random

Foi implementado também a geração dinâmica de problemas, isto é, o jogo implementado gera aleatoriamente o tabuleiro a ser resolvido. Neste modo de jogo, o utilizador insere o tamanho do tabuleiro que pretende bem como as posições inicial e final da cobra. Com estes dados, é gerado um tabuleiro com as pistas necessárias para a sua resolução, como o exemplo seguinte, com dimensão 15x15:



Para o modo *random* são utilizadas as mesmas restrições e após ter uma solução do tabuleiro é eliminada a cobra e são inseridas no tabuleiro as pistas. O número de pistas a apresentar é consoante o tamanho do tabuleiro, sendo calculado pela operação `tamanhoTab div 2`.

4 Visualização da Solução

Quando o jogo é iniciado podemos observar um menu inicial:

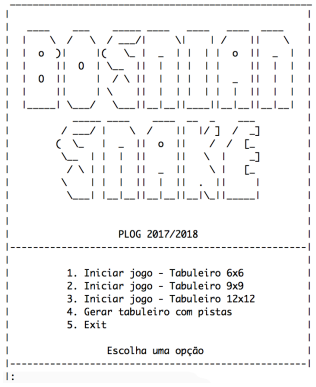


Figura 3 Menu inicial

Este menu é desenhado pelo predicado *imprimirMenuInicial.* Podemos observar que existem 5 opções no menu, estas são selecionadas pelo predicado *selecionarTabuleiro*, que por sua vez recorre ao predicado *modoJogo* para iniciar o modo de jogo desejado. Se a opção introduzida não corresponder a nenhuma das opções disponíveis o predicado volta a pedir uma opção válida.

```
selecionarTabuleiro:-
        read(OPCAO),
        OPCAO > 0,
        OPCAO < 6, !,
        modoJogo(OPCAO).

selecionarTabuleiro:-
        write('Introduza uma opção válida!'),nl,
        selecionarTabuleiro.

modoJogo(1):-
        iniciarJogo(6).

modoJogo(2):-
        iniciarJogo(9).

modoJogo(3):-
        iniciarJogo(12).

modoJogo(4):-
         iniciarJogoRandom.

modoJogo(5):-
         write('Exit!').
```

Excerto de código 8 Imprimir menu inicial e opções

O tabuleiro de jogo tem 3 tamanhos disponíveis, 6x6, 9x9 e 12x12 como mostram as figuras seguintes. A cabeça, cauda e corpo da cobra são representados pelo caracter "X". Quando é iniciado o tabuleiro de jogo, é possível observar dois caracteres "X" que representam a cabeça e a cauda da cobra, e alguns números que representam as pistas.

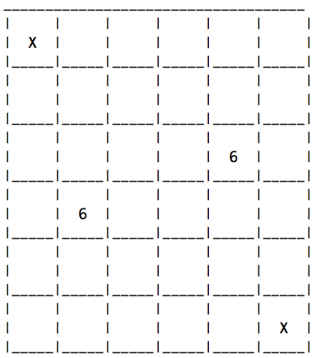


Figura 4 Exemplo jogo 6x6

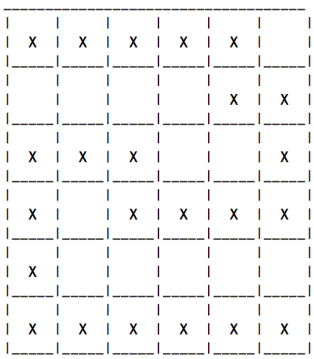


Figura 5 Possível resolução jogo 6x6

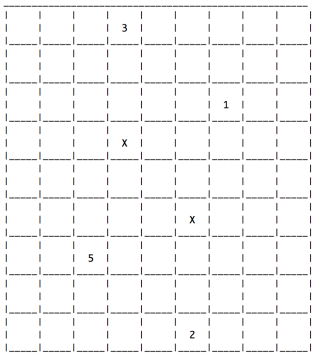


Figura 6 Exemplo jogo 9x9

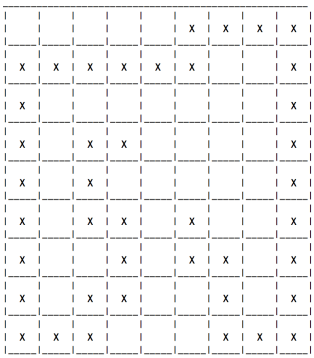


Figura 7 Possível resolução jogo 9x9

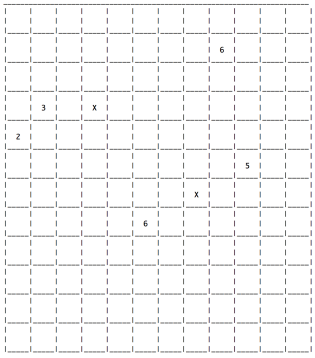


Figura 8 Exemplo de jogo 12x12

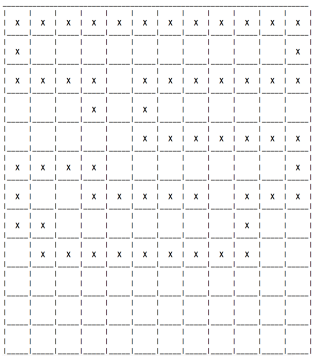


Figura 9 Possível resolução de jogo 12x12

Estes tabuleiros são desenhados com o auxílio dos predicados recursivos e adaptáveis a qualquer dimensão do tabuleiro:

- *imprimirSeparadorInicial* – imprime os caracteres "_____", para formar a barra do topo do tabuleiro;
- *imprimirSeparadorLinhas* – imprime os caracteres "|_____"; para criar um pedaço da célula do tabuleiro;
- *imprimirSeparadorColunas* – imprime os carateres "| " para criar o espaço necessário entre as células que não estão ocupadas por uma peça;
- *imprimirCasa* – imprime a célula com a peça, representada pelo caracter "X";
- *imprimirPecasLinha* – imprime as peças que estão numa determinada linha;
- *imprimirLinha* – imprime a linha completa;
- *imprimirLinhas* – imprime o tabuleiro completo;
- *imprimirTabuleiro* – predicado que chama os predicados necessários para a impressão completa do tabuleiro;

```
imprimirSeparadorInicial(0,_).
imprimirSeparadorInicial(TAMANHO,SEPARADOR):-
        write(SEPARADOR), TAMANHO1 is TAMANHO - 1,
        imprimirSeparadorInicial(TAMANHO1, SEPARADOR).
imprimirSeparadorInicial(TAMANHO):-
        imprimirSeparadorInicial(TAMANHO,'_____').
imprimirSeparadorLinhas(0,_).
imprimirSeparadorLinhas(TAMANHO,SEPARADOR):-
        write(SEPARADOR),
        TAMANHO1 is TAMANHO - 1,
        imprimirSeparadorLinhas(TAMANHO1, SEPARADOR).
imprimirSeparadorLinhas(TAMANHO):-
        imprimirSeparadorLinhas(TAMANHO,'|_____').

imprimirSeparadorColunas(0,_).
imprimirSeparadorColunas(TAMANHO,SEPARADOR):-
        write(SEPARADOR),
        TAMANHO1 is TAMANHO - 1,
        imprimirSeparadorColunas(TAMANHO1, SEPARADOR).
imprimirSeparadorColunas(TAMANHO):-
        imprimirSeparadorColunas(TAMANHO,'|     ').
imprimirBarraSeparadorColuna:-
        write('|'), nl.

imprimirCasa(0):-
        write('     ').
imprimirCasa(-1):-
        write('  '), write('X'), write('  ').
imprimirCasa(H):-
        write('  '), write(H), write('  ').

imprimirPecasLinha([]).
imprimirPecasLinha([H | T]):-
        write('|'),
        imprimirCasa(H),
        imprimirPecasLinha(T).

imprimirLinha([],_).
```

```
imprimirLinha(LINHA,TAMANHO):-
        imprimirSeparadorColunas(TAMANHO),
        imprimirBarraSeparadorColuna,
        imprimirPecasLinha(LINHA),
        imprimirBarraSeparadorColuna,
        imprimirSeparadorLinhas(TAMANHO),
        imprimirBarraSeparadorColuna.

imprimirLinhas([],_).
imprimirLinhas([LINHA | T],TAMANHO):-
        imprimirLinha(LINHA,TAMANHO),
        imprimirLinhas(T,TAMANHO).

imprimirTabuleiro(TAB):-
        getTamanhoTab(TAB,TAMANHO),
        imprimirSeparadorInicial(TAMANHO), nl,
        imprimirLinhas(TAB,TAMANHO).
```

Excerto de código 9 Código que imprime o tabuleiro

5 Resultados

Antes de elaborar a análise, iremos apresentar os termos mencionados na Tabela 1:

- Retomadas – quando a execução de um predicado é interrompida pela necessidade de recorrer a outro predicado auxiliar e, após a execução do segundo continua a correr o primeiro predicado;
- Envolvimentos -
- Podas – soluções obtidas, mas não são relevantes para a continuação do problema e por esse motivo são descartadas. Por outras palavras, corta soluções extra da árvore;
- Retrocessos – backtracking;
- Restrições criadas – todas as restrições criadas durante o programa;
- Tempo – tempo decorrido desde que o programa inicia até que chega ao fim;
- Tempo inverso – tem o mesmo significado de que o primeiro, mas o *labeling* corre de modo inverso.

Dados | Tabuleiro	6x6	9x9	12x12
Retomadas	5172895	5198062	5183971
Envolvimentos	411436	436226	419726
Podas	10859449	10875492	10868193
Retrocessos	1199160	1199186	1199167
Restrições criadas	1438	3686	7039

Tabela 1

		Tempo (s)	Tempo inverso (s)
Tabuleiro 6x6	Exemplo 1	6.46	6.75
	Exemplo 2	6.34	6.56

	Exemplo 3	6.32	6.75
	Média	**6.37**	**6.60**
Tabuleiro 9x9	Exemplo 1	6.60	6.41
	Exemplo 2	6.30	6.38
	Exemplo 3	6.12	6.38
	Média	**6.34**	**6.39**
Tabuleiro 12x12	Exemplo 1	6.42	6.58
	Exemplo 2	6.29	6.46
	Exemplo 3	6.02	6.32
	Média	**6.24**	**6.45**

Tabela 2

Através da tabela anterior é possível concluir que o tabuleiro mais eficiente em Tempo é o tabuleiro 12x12 e o tabuleiro mais eficiente em Tempo inverso é o tabuleiro 9x9. No entanto, quando observamos a Tabela 1 é possível perceber que os tabuleiros 6x6 e 12x12 têm valores bastante idênticos em Retomadas, Envolvimentos, Podas e Retrocessos. Em conclusão, se for dado tempo suficiente para o algoritmo correr, acabar-se-á por chegar a uma solução ótima.

6 Conclusões e Trabalho Futuro

Após a realização deste trabalho, concluímos que a linguagem de Programação em Lógica, mais especificamente os módulos de resolução de restrições são bastante úteis, permitindo a resolução de uma ampla variedade de questões de decisões e otimização.

Depois de entendido o funcionamento por de trás de variáveis de decisão, formas de restringir o domínio de variáveis, a maneira como o *labeling* funciona e, por fim, o todo o potencial da biblioteca *clfpd* disponibiliza, tornou-se mais acessível uma resolução para o problema.

O grupo sentiu mais dificuldade na implementação de funcionalidades random com alguma eficiência.

A solução implementada pelo nosso grupo corresponde às expectativas em geral.

Bibliografia

1. http://logicmastersindia.com/lmitests/dl.asp?attachmentid=645&view=1
2. . https://sicstus.sics.se/

Anexo

Código fonte: main.pl

```
:-use_module(library(random)).
:-ensure_loaded('impressao.pl').
:-ensure_loaded('geral.pl').
:-ensure_loaded('logica.pl').
:-ensure_loaded('random.pl').

/* ******
 * main *
 ****** */

% funcao principal, ou seja, primrira funcao a ser invocada para iniciar o
jogo
run:-
        imprimirMenuInicial,
        selecionarTabuleiro.

% opcoes de modo de jogo
% opcao 1 - Iniciar jogo - Tabuleiro 6x6
modoJogo(1):-
        iniciarJogo(6).

% opcao 2 - Iniciar jogo - Tabuleiro 9x9
modoJogo(2):-
        iniciarJogo(9).

% opcao 3 - Iniciar jogo - Tabuleiro 12x12
modoJogo(3):-
        iniciarJogo(12).

% opcao 4 - Gerar tabuleiro com pistas
modoJogo(4):-
         iniciarJogoRandom.

% opcao 5 - exit
modoJogo(5):-
         write('Exit!').

% funcao para iniciar o jogo
iniciarJogo(TAMANHO):-
        tabuleiro(TAMANHO,TAB),
        getCasasCobraLinhaColuna(TAMANHO,COBRA_LINHA,COBRA_COLUNA),
        imprimirTabuleiro(TAB),
        getSolucao(TAB,COBRA_LINHA,COBRA_COLUNA,TAB1), !,
        imprimirTabuleiro(TAB1).

% Obtem a solucao de um tabuleiro
getSolucao(TAB,COBRA_LINHA,COBRA_COLUNA,TAB1):-
        getTamanhoTab(TAB,TAMANHO), !,
        validarDuasPosicoesCobra(TAB,POSICOES,TAMANHO),
        inicializarTab(TAB,TAB1), % definir domínios
        validarInicioFimCobra(TAB1,POSICOES), % 1ª restrição
        validarCasasCobraLinhaColuna(TAB1,COBRA_LINHA,COBRA_COLUNA), % 2ª
restrição
        validarCasasCobraAround(TAB,TAB1,1), % 3ª restrição
        validarDoisVizinhosCobra(TAB1,TAB1,1,POSICOES), % 4ª restrição
        validarIntersecaoDiagonal(TAB1,TAB1,TAMANHO), % 5ª restrição
```

```
        maplist(labeling([]),TAB1),
        verificarCobraConexa(TAB1,TAMANHO,POSICOES).

% Jogo Random
% lê o tamanho do tabuleiro da opcao 4
introduzirTamanhoTabuleiro(TAMANHO):-
        write('Introduza um tamanho para o tabuleiro valido!'),nl,
        read(TAMANHO),nl,
        TAMANHO > 2, !.

introduzirTamanhoTabuleiro(TAMANHO):-
        introduzirTamanhoTabuleiro(TAMANHO).

% lê a posição inicial e final da snake no tabuleiro com random
introduzirPosicaoInicialFinal(TAB,TAMANHO,POSICOES,TAB1):-
        write('Introduza uma posicao inicial e final para a cobra no for-
mato:'),nl,
        write('LinhaInicial/ColunaInicial - LinhaFinal/ColunaFinal'),nl,
        read(LI/CI - LF/CF),nl,
        inicializarTabuleiroRandom(TAB,LI/CI,LF/CF,TAB1),
        validarDuasPosicoesCobra(TAB1,POSICOES,TAMANHO).

introduzirPosicaoInicialFinal(TAB,TAMANHO,[LI/CI,LF/CF],TAB1):-
        introduzirPosicaoInicialFinal(TAB,TAMANHO,[LI/CI,LF/CF],TAB1).

% gerador de pistas
gerarPistas(_,0,_,_,PISTAS,PISTAS).

gerarPistas(TAB,NPISTAS,MAX,POSICOES,AUXPISTAS,PISTAS):-
        random(1,MAX,LINHA),
        random(1,MAX,COLUNA),
        \+member(LINHA/COLUNA,POSICOES),
        getCasaTabuleiro(TAB,LINHA,COLUNA,CASA),
        CASA = 0,
        getElementosVizinhos(TAB,[[-1,-1],[-1,0],[-1,1],[0,-1],[0,1],[1,-
1],[1,0],[1,1]],LINHA,COLUNA,LISTAVIZINHOS),
        sumlist(LISTAVIZINHOS,SUM),
        SUM < 0,
        NPISTAS1 is NPISTAS - 1,
        gerarPis-
tas(TAB,NPISTAS1,MAX,POSICOES,[LINHA/COLUNA|AUXPISTAS],PISTAS).

gerarPistas(TAB,NPISTAS,MAX,POSICOES,AUXPISTAS,PISTAS):-
        gerarPistas(TAB,NPISTAS,MAX,POSICOES,AUXPISTAS,PISTAS).

% apaga a cobra
apagarCobraEscreverPistasLinha(_,[],[],_,_,_,_).

apagarCobraEscreverPistasLi-
nha(TAB,[_|T],[SUM1|T1],PISTAS,POSICOES,NLINHA,NCOLUNA):-
        member(NLINHA/NCOLUNA,PISTAS),
        getElementosVizinhos(TAB,[[-1,-1],[-1,0],[-1,1],[0,-1],[0,1],[1,-1],
[1,0],[1,1]],NLINHA,NCOLUNA,LISTAVIZINHOS),
        sumlist(LISTAVIZINHOS,SUM),
        SUM1 is abs(SUM),
        NCOLUNA1 is NCOLUNA + 1,
        apagarCobraEscreverPistasLinha(TAB,T,T1,PISTAS,POSICOES,NLINHA,
NCOLUNA1).

apagarCobraEscreverPistasLinha(TAB,[_|T],[-1|T1],PISTAS,POSICOES,NLINHA,
NCOLUNA):-
        member(NLINHA/NCOLUNA,POSICOES),
        NCOLUNA1 is NCOLUNA + 1,
```

```
        apagarCobraEscreverPistasLinha(TAB,T,T1,PISTAS,POSICOES,NLINHA,
NCOLUNA1).

apagarCobraEscreverPistasLinha(TAB,[_|T],[0|T1],PISTAS,POSICOES,NLINHA,
NCOLUNA):-
        NCOLUNA1 is NCOLUNA + 1,
        apagarCobraEscreverPistasLinha(TAB,T,T1,PISTAS,POSICOES,NLINHA,
NCOLUNA1).

apagarCobraEscreverPistas(_,[],[],_,_,_).

apagarCobraEscreverPistas(TAB,[H|T],[H1|T1],PISTAS,POSICOES,NLINHA):-
        apagarCobraEscreverPistasLinha(TAB,H,H1,PISTAS,POSICOES,NLINHA,1),
        NLINHA1 is NLINHA + 1,
        apagarCobraEscreverPistas(TAB,T,T1,PISTAS,POSICOES,NLINHA1).

apagarCobraEscreverPistas(TAB,TAMANHO,POSICOES,TAB1):-
        NPISTAS is TAMANHO div 2,
        MAX is TAMANHO + 1,
        gerarPistas(TAB,NPISTAS,MAX,POSICOES,[],PISTAS),
        apagarCobraEscreverPistas(TAB,TAB,TAB1,PISTAS,POSICOES,1).

% inicia o jogo random
iniciarJogoRandom(TAMANHO,POSICOES,TAB):-
        imprimirTabuleiro(TAB),
        inicializarTab(TAB,TAB1),
        validarInicioFimCobra(TAB1,POSICOES),
        validarCasasCobraAround(TAB,TAB1,1),
        validarDoisVizinhosCobra(TAB1,TAB1,1,POSICOES),
        validarIntersecaoDiagonal(TAB1,TAB1,TAMANHO),
        maplist(labeling([]),TAB1),
        verificarCobraConexa(TAB1,TAMANHO,POSICOES),
        imprimirTabuleiro(TAB1),
        apagarCobraEscreverPistas(TAB1,TAMANHO,POSICOES,TAB2),
        imprimirTabuleiro(TAB2).

iniciarJogoRandom:-
        introduzirTamanhoTabuleiro(TAMANHO),
        gerarTabuleiro(TAMANHO,TAB),
        introduzirPosicaoInicialFinal(TAB,TAMANHO,POSICOES,TAB1),
        iniciarJogoRandom(TAMANHO,POSICOES,TAB1).
```

Código fonte: impressao.pl

```
/* ***********
 * impressao *
 *********** */

imprimirMenuInicial:-
        nl,nl,
        write(' _____________________________________________________'), nl,
        write('|                                                     |'), nl,
        write('|   ____    ___   _____ ____   ____   ____  ____     |'), nl,
        write('|  |    \\  /   \\  / ___/|     \\||    | /    ||    \\   |'), nl,
        write('|  |  o  )|     |(   \\_ |  _  | |  | |  o  ||  _  |  |'), nl,
        write('|  |     ||  O  | \\__  ||  |  | |  | |     ||  |  |  |'), nl,
        write('|  |  O  ||     |  / \\ ||  |  | |  | |  _  ||  |  |  |'), nl,
        write('|  |     ||     | \\    ||  |  | |  | |  |  ||  |  |  |'), nl,
        write('|  |_____| \\___/   \\___||__|__||____||__|__||__|__|  |'), nl,
        write('|            _____ ____   ____  __  _    ___         |'), nl,
        write('|          / ___/ |    \\  /    ||  |/ ]  /  _]        |'), nl,
        write('|         (  \\_   |  _  ||  o  ||    /  /  [_         |'), nl,
        write('|          \\__  | |  |  ||     ||   \\  |    _]        |'), nl,
        write('|          / \\ | |  |  ||  _  ||    \\ |   [_         |'), nl,
        write('|          \\    | |  |  ||  |  ||  .  ||     |        |'), nl,
        write('|           \\___| |__|__||__|__||__|\\_||_____|        | '), nl,
         write('|                                                     |'), nl,
         write('|                                                     |'), nl,
         write('|                    PLOG 2017/2018                   |'), nl,
         write('|-----------------------------------------------------|'), nl,
         write('|                                                     |'), nl,
         write('|          1. Iniciar jogo - Tabuleiro 6x6            |'), nl,
         write('|          2. Iniciar jogo - Tabuleiro 9x9            |'), nl,
         write('|          3. Iniciar jogo - Tabuleiro 12x12          |'), nl,
         write('|          4. Gerar tabuleiro com pistas              |'), nl,
         write('|          5. Exit                                    |'), nl,
         write('|                                                     |'), nl,
         write('|                 Escolha uma opção                   |'), nl,
         write('|-----------------------------------------------------|'), nl.

selecionarTabuleiro:-
        read(OPCAO),
        OPCAO > 0,
        OPCAO < 6, !,
        modoJogo(OPCAO).

selecionarTabuleiro:-
        write('Introduza uma opção válida!'),nl, selecionarTabuleiro.

% Imprime o limite superior do tabuleiro
imprimirSeparadorInicial(0,_).

imprimirSeparadorInicial(TAMANHO,SEPARADOR):-
        write(SEPARADOR), TAMANHO1 is TAMANHO - 1,
        imprimirSeparadorInicial(TAMANHO1, SEPARADOR).

imprimirSeparadorInicial(TAMANHO):-
        imprimirSeparadorInicial(TAMANHO,'______').
```

```
% Imprime o separador de linhas do tabuleiro
imprimirSeparadorLinhas(0,_).

imprimirSeparadorLinhas(TAMANHO,SEPARADOR):-
        write(SEPARADOR), TAMANHO1 is TAMANHO - 1,
        imprimirSeparadorLinhas(TAMANHO1, SEPARADOR).

imprimirSeparadorLinhas(TAMANHO):-
        imprimirSeparadorLinhas(TAMANHO,'|_____').

% Imprime o separador de colunas do tabuleiro
imprimirSeparadorColunas(0,_).

imprimirSeparadorColunas(TAMANHO,SEPARADOR):-
        write(SEPARADOR), TAMANHO1 is TAMANHO - 1,
        imprimirSeparadorColunas(TAMANHO1, SEPARADOR).

imprimirSeparadorColunas(TAMANHO):-
        imprimirSeparadorColunas(TAMANHO,'|     ').

imprimirBarraSeparadorColuna:- write('|'), nl.

% Imprime uma casa do tabuleiro com a peca "Peca"
imprimirCasa(0):- write('     ').
imprimirCasa(-1):- write('  '), write('X'), write('  ').
imprimirCasa(H):- write('  '), write(H), write('  ').

% Imprime as pecas que estao numa determinada linha do tabuleiro
imprimirPecasLinha([]).
imprimirPecasLinha([H | T]):-
        write('|'),
        imprimirCasa(H),
        imprimirPecasLinha(T).

% Imprime a linha numero "NLinha" do tabuleiro
imprimirLinha([],_).
imprimirLinha(LINHA,TAMANHO):-
        imprimirSeparadorColunas(TAMANHO),
        imprimirBarraSeparadorColuna,
        imprimirPecasLinha(LINHA),
        imprimirBarraSeparadorColuna,
        imprimirSeparadorLinhas(TAMANHO),
        imprimirBarraSeparadorColuna.

% Imprime todas as linhas do tabuleiro
imprimirLinhas([],_).
imprimirLinhas([LINHA | T],TAMANHO):-
        imprimirLinha(LINHA,TAMANHO),
        imprimirLinhas(T,TAMANHO).

% Imprime o tabuleiro com o estado atual do jogo
imprimirTabuleiro(TAB):-
        getTamanhoTab(TAB,TAMANHO),
        imprimirSeparadorInicial(TAMANHO), nl,
        imprimirLinhas(TAB,TAMANHO).
```

Código fonte: geral.pl

```
/**********
 tabuleiros
**********/

tabuleiro(6,
[[-1,0,0,0,0,0], [0,0,0,0,0,0], [0,0,0,0,6,0],
 [0,6,0,0,0,0], [0,0,0,0,0,0], [0,0,0,0,0,-1]]).

tabuleiro(9,
[[0,0,0,3,0,0,0,0,0], [0,0,0,0,0,0,0,0,0], [0,0,0,0,0,0,1,0,0],
 [0,0,0,-1,0,0,0,0,0], [0,0,0,0,0,0,0,0,0], [0,0,0,0,0,-1,0,0,0],
 [0,0,5,0,0,0,0,0,0], [0,0,0,0,0,0,0,0,0], [0,0,0,0,0,2,0,0,0]]).

tabuleiro(12,
[[0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,6,0,0,0],[0,0,0,0,0,0,0,0,0,0,
0],
[0,3,0,-
1,0,0,0,0,0,0,0,0],[2,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,5,0,0],
[0,0,0,0,0,0,0,0,-
1,0,0,0,0],[0,0,0,0,0,0,6,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0
]]).


% tablueiro(TAMANHO,LINHA,CASAS_COBRA_LINHA)
tabuleiroLinha(6,2,2).
tabuleiroLinha(6,5,1).

tabuleiroLinha(12,9,9).
tabuleiroLinha(12,4,2).

% tablueiro(TAMANHO,COLUNA,CASAS_COBRA_LINHA)
tabuleiroColuna(9,3,6).
tabuleiroColuna(9,7,4).

tabuleiroColuna(12,8,5).
tabuleiroColuna(12,2,5).
```

Código fonte: logica.pl

```
:-use_module(library(clpfd)).
:-use_module(library(lists)).
:-use_module(library(between)).

/***************
 logica do jogo
***************/

% Quinta Restrição - Nao pode haver diagonais

validarDiagonalNaoDiagonalRestricao(TAB,NLINHA,NCOLUNA,NLINHA1,NCOLUNA1):-
        getCasaTabuleiro(TAB,NLINHA,NCOLUNA,CASA),
        getCasaTabuleiro(TAB,NLINHA1,NCOLUNA1,CASA1),
        getCasaTabuleiro(TAB,NLINHA1,NCOLUNA,CASAVIZINHA1),
        getCasaTabuleiro(TAB,NLINHA,NCOLUNA1,CASAVIZINHA2),
        (CASA #= -1 #/\ CASA1 #= -1) #=>
        (CASAVIZINHA1 #= -1 #/\ CASAVIZINHA2 #= 0)
        #\/
        (CASAVIZINHA1 #= 0 #/\ CASAVIZINHA2 #= -1).

validarDiagonalNaoDiagonal(_,[],_,_,_).

validarDiagonalNaoDiagonal(TAB,[[NL,NC]|T],NLINHA,NCOLUNA,TAMANHO):-
        NLINHA1 is NLINHA - NC,
        NCOLUNA1 is NCOLUNA + NL,
        between(1,TAMANHO,NLINHA1),
        between(1,TAMANHO,NCOLUNA1),
        validarDiagonalNaoDiagonalRestri-
cao(TAB,NLINHA,NCOLUNA,NLINHA1,NCOLUNA1), !,
        validarDiagonalNaoDiagonal(TAB,T,NLINHA,NCOLUNA,TAMANHO).

validarDiagonalNaoDiagonal(TAB,[[_,_]|T],NLINHA,NCOLUNA,TAMANHO):-
        validarDiagonalNaoDiagonal(TAB,T,NLINHA,NCOLUNA,TAMANHO).

validarIntersecaoDiagonal([],_,_,_,_).

validarIntersecaoDiagonal([_|T],TAB1,NLINHA,NCOLUNA,TAMANHO):-
        validarDiagonalNaoDiagonal(TAB1,[[-1,-1],[-1,1],[1,-
1],[1,1]],NLINHA,NCOLUNA,TAMANHO),
        NCOLUNA1 is NCOLUNA + 1,
        validarIntersecaoDiagonal(T,TAB1,NLINHA,NCOLUNA1,TAMANHO).

validarIntersecaoDiagonal([],_,_,_).

validarIntersecaoDiagonal([H|T],TAB1,NLINHA,TAMANHO):-
        validarIntersecaoDiagonal(H,TAB1,NLINHA,1,TAMANHO),
        NLINHA1 is NLINHA + 1,
        validarIntersecaoDiagonal(T,TAB1,NLINHA1,TAMANHO).

validarIntersecaoDiagonal(TAB,TAB1,TAMANHO):-
        validarIntersecaoDiagonal(TAB,TAB1,1,TAMANHO).

% Quarta Restrição - A cobra não pode tocar em si mesma

validarDoisVizinhosCobra([],_,_,_,_).

validarDoisVizinhosCobra([_|T],TAB,NLINHA,NCOLUNA,[LI/CI,LF/CF]):-
        ((NLINHA = LF, NCOLUNA = CF) ; (NLINHA = LI, NCOLUNA = CI)),
        NCOLUNA1 is NCOLUNA + 1,
        validarDoisVizinhosCobra(T,TAB,NLINHA,NCOLUNA1,[LI/CI,LF/CF]),!.
```

```
validarDoisVizinhosCobra([_|T],TAB,NLINHA,NCOLUNA,[LI/CI,LF/CF]):-
        getElementosVizinhos(TAB,[[-1,0],[0,1],[1,0],[0,-
1]],NLINHA,NCOLUNA,LISTAVIZINHOS),
        getCasaTabuleiro(TAB,NLINHA,NCOLUNA,CASA),
        count(-1,LISTAVIZINHOS,#=,COUNT),
        CASA #= -1 #=> COUNT #= 2,
        NCOLUNA1 is NCOLUNA + 1,
        validarDoisVizinhosCobra(T,TAB,NLINHA,NCOLUNA1,[LI/CI,LF/CF]).

validarDoisVizinhosCobra([],_,_,_).

validarDoisVizinhosCobra([H|T],TAB1,NLINHA,POSINICIALFINAL):-
        validarDoisVizinhosCobra(H,TAB1,NLINHA,1,POSINICIALFINAL),
        NLINHA1 is NLINHA + 1,
        validarDoisVizinhosCobra(T,TAB1,NLINHA1,POSINICIALFINAL).

% Terceira Restrição - O número de casa ocupadas pela cobra em volta da casa
'H' tem que ser igual a H (H e um numero de 1 a 7)

validarCasasCobraAroundLinha([],_,_,_).

validarCasasCobraAroundLinha([H|T],TAB1,NLINHA,NCOLUNA):-
        H > 0,
        getElementosVizinhos(TAB1,[[-1,-1],[-1,0],[-1,1],[0,-1],[0,1],[1,-
1],[1,0],[1,1]],NLINHA,NCOLUNA,LISTAVIZINHOS),
        getCasaTabuleiro(TAB1,NLINHA,NCOLUNA,CASA),
        count(-1,LISTAVIZINHOS,#=,COUNT),
        CASA #= 0 #=> COUNT #= H,
        NCOLUNA1 is NCOLUNA + 1,
        validarCasasCobraAroundLinha(T,TAB1,NLINHA,NCOLUNA1).

validarCasasCobraAroundLinha([_|T],TAB1,NLINHA,NCOLUNA):-
        NCOLUNA1 is NCOLUNA + 1,
        validarCasasCobraAroundLinha(T,TAB1,NLINHA,NCOLUNA1).

validarCasasCobraAround([],_,_).

validarCasasCobraAround([H|T],TAB1,NLINHA):-
        validarCasasCobraAroundLinha(H,TAB1,NLINHA,1), % H e a linha, TAB1 e
o tabueleiro solucao, TAMANHO e o tamanho do tabuleiro, NLINHA e o numero da
linha
        NLINHA1 is NLINHA + 1,
        validarCasasCobraAround(T,TAB1,NLINHA1).

% Segunda Restrição - O total de casas ocupadas pela cobra numa linha ou co-
luna tem que ser igual a N

validarCasasCobraColuna(_,[]).

validarCasasCobraColuna(TAB,[COLUNA-COBRACOLUNA|T]):-
        nth1(COLUNA,TAB,COLUNA_TAB),
        sum(COLUNA_TAB,#=,COBRACOLUNA1),
        COBRACOLUNA1 #= (COBRACOLUNA * -1),
        validarCasasCobraColuna(TAB,T).

validarCasasCobraLinha(_,[]).

validarCasasCobraLinha(TAB,[LINHA-COBRALINHA|T]):-
        nth1(LINHA,TAB,LINHA_TAB),
        sum(LINHA_TAB,#=,COBRALINHA1),
        COBRALINHA1 #= (COBRALINHA * -1),
        validarCasasCobraColuna(TAB,T).

validarCasasCobraLinhaColuna(TAB,COBRA_LINHA,COBRA_COLUNA):-
```

```
        validarCasasCobraLinha(TAB,COBRA_LINHA),
        transpose(TAB,TTAB),
        validarCasasCobraColuna(TTAB,COBRA_COLUNA).

% Primeira Restrição - O inicio e o fim da cobra so pode ter um vizinho

% Percorre-se coluna a coluna porque o inicio e fim da cobra pode estar na
mesma linha

validarInicioFimCobra(TAB,NLINHA,NCOLUNA):-
        getElementosVizinhos(TAB,[[-1,0],[0,1],[1,0],[0,-
1]],NLINHA,NCOLUNA,LISTAVIZINHOS),
        getCasaTabuleiro(TAB,NLINHA,NCOLUNA,CASA),
        count(-1,LISTAVIZINHOS,#=,COUNT),
        CASA #= -1 #=> COUNT #= 1.

validarInicioFimCobra(_,[]).

validarInicioFimCobra(TAB,[L/C|T]):-
        validarInicioFimCobra(TAB,L,C),
        validarInicioFimCobra(TAB,T).

% Define o domínio das casas do tabuleiro solucao

inicializarLinha([],[]).

inicializarLinha([-1|T],[H1|T1]):-
        H1 in -1.. -1,
        inicializarLinha(T,T1).

inicializarLinha([H|T],[H1|T1]):-
        H > 0,
        H1 in 0..0,
        inicializarLinha(T,T1).

inicializarLinha([_|T],[H1|T1]):-
        H1 in -1..0,
        inicializarLinha(T,T1).

inicializarTab([],[]).

inicializarTab([HTAB|TTAB],[HTAB1|TTAB1]):-
        inicializarLinha(HTAB,HTAB1),
        inicializarTab(TTAB,TTAB1).

% Valida se no tabuleiro inicial existem apenas dois zeros (inicio e fim da
cobra)

validarDuasPosicoesCobra([],_,_,POSICOES,POSICOES).

validarDuasPosicoesCobra([H|T],NLINHA,NCOLUNA,POSICOES,POSICOES1):-
        H = -1,
        NCOLUNA1 is NCOLUNA + 1,
        validarDuasPosicoesCo-
bra(T,NLINHA,NCOLUNA1,[NLINHA/NCOLUNA|POSICOES],POSICOES1).

validarDuasPosicoesCobra([_|T],NLINHA,NCOLUNA,POSICOES,POSICOES1):-
        NCOLUNA1 is NCOLUNA + 1,
        validarDuasPosicoesCobra(T,NLINHA,NCOLUNA1,POSICOES,POSICOES1).

validarDuasPosicoesCobra([],_,POSICOES,POSICOES).

validarDuasPosicoesCobra([H|T],NLINHA,POSICOES,POSICOES2):-
        validarDuasPosicoesCobra(H,NLINHA,1,POSICOES,POSICOES1),
```

```
        NLINHA1 is NLINHA + 1,
        validarDuasPosicoesCobra(T,NLINHA1,POSICOES1,POSICOES2).

validarDuasPosicoesCobra(TAB,[LI/CI,LF/CF],TAMANHO):-
        validarDuasPosicoesCobra(TAB,1,[],[LI/CI,LF/CF]),
        getListaComPosicoes([[-1,0],[0,1],[1,0],[0,-
1]],LI,CI,TAMANHO,LISTAVIZINHOS),
        \+member(LF-CF,LISTAVIZINHOS).

/******
 utils
*******/

getListaComPosicoes([],_,_,_,[]).

getListaComPosicoes([[L,C]|T],NLINHA,NCOLUNA,TAMANHO,[LINHA1-
COLUNA1|POSICOES]):-
        LINHA1 is NLINHA + L,
        between(1,TAMANHO,LINHA1),
        COLUNA1 is NCOLUNA + C,
        between(1,TAMANHO,COLUNA1), !,
        getListaComPosicoes(T,NLINHA,NCOLUNA,TAMANHO,POSICOES).

getListaComPosicoes([_|T],NLINHA,NCOLUNA,TAMANHO,POSICOES):-
        getListaComPosicoes(T,NLINHA,NCOLUNA,TAMANHO,POSICOES).

getListaComElementos(_,[],[]).

getListaComElementos(TAB,[NLINHA-NCOLUNA|T],[ELMEN|ELEMENTOS]):-
        getElemento(TAB,NLINHA,NCOLUNA,ELMEN),
        getListaComElementos(TAB,T,ELEMENTOS).

getElementosVizinhos(TAB,POSICOES,NLINHA,NCOLUNA,ELEMENTOS):-
        getTamanhoTab(TAB,TAMANHO),
        getListaComPosicoes(POSICOES,NLINHA,NCOLUNA,TAMANHO,LISTAVIZINHOS),
        getListaComElementos(TAB,LISTAVIZINHOS,ELEMENTOS).

getElemento(TAB,NLINHA,NCOLUNA,ELEMENTO):-
        getCasaTabuleiro(TAB,NLINHA,NCOLUNA,ELEMENTO).

getTamanhoTab([H|_],TAMANHO):-
        length(H,TAMANHO).

getCasaTabuleiro(TAB,NLINHA,NCOLUNA,CASA):-
        nth1(NLINHA,TAB,LINHA_TAB),
        nth1(NCOLUNA,LINHA_TAB,CASA).

getCasasCobraLinhaColuna(TAMANHO,COBRA_LINHA,COBRA_COLUNA):-
        findall(
                LINHA-COBRALINHA,
                tabuleiroLinha(TAMANHO,LINHA,COBRALINHA),
                COBRA_LINHA),
        findall(
                COLUNA-COBRALINHA,
                tabuleiroColuna(TAMANHO,COLUNA,COBRALINHA),
                COBRA_COLUNA).

% Verifica conexão de todos os -1 da solução

getVizinhoConexo(TAB,[NLINHA-NCOLUNA|_],NLINHA,NCOLUNA,POSICOES):-
        getCasaTabuleiro(TAB,NLINHA,NCOLUNA,CASA),
        CASA = -1,
        \+member(NLINHA/NCOLUNA,POSICOES), !.
```

```
getVizinhoConexo(TAB,[_|T],NLINHA,NCOLUNA,POSICOES):-
        getVizinhoConexo(TAB,T,NLINHA,NCOLUNA,POSICOES).

getListaPosicoesCobra(_,LF/CF,LF/CF,_,POSICOESCOBRA,POSICOESCOBRA).

getListaPosicoesCobra(TAB,LI/CI,LF/CF,TAMANHO,POSICOES,POSICOESCOBRA):-
        getListaComPosicoes([[-1,0],[0,1],[1,0],[0,-
1]],LI,CI,TAMANHO,LISTAVIZINHOS),
        getVizinhoConexo(TAB,LISTAVIZINHOS,NLINHA,NCOLUNA,POSICOES),
        getListaPosicoesCo-
bra(TAB,NLINHA/NCOLUNA,LF/CF,TAMANHO,[NLINHA/NCOLUNA|POSICOES],POSICOESCOBRA
).

getListaPosicoesCobra(TAB,TAMANHO,[PI,PF],POSICOESCOBRA):-
        getListaPosicoesCobra(TAB,PI,PF,TAMANHO,[PI],POSICOESCOBRA).

validarTodosMenosUmEmCobraLinha([],_,_,_).

validarTodosMenosUmEmCobraLinha([H|T],POSICOESCOBRA,NLINHA,NCOLUNA):-
        H = -1,
        member(NLINHA/NCOLUNA,POSICOESCOBRA),
        NCOLUNA1 is NCOLUNA + 1,
        validarTodosMenosUmEmCobraLinha(T,POSICOESCOBRA,NLINHA,NCOLUNA1).

validarTodosMenosUmEmCobraLinha([H|T],POSICOESCOBRA,NLINHA,NCOLUNA):-
        H = 0,
        NCOLUNA1 is NCOLUNA + 1,
        validarTodosMenosUmEmCobraLinha(T,POSICOESCOBRA,NLINHA,NCOLUNA1).

validarTodosMenosUmEmCobra([],_,_).

validarTodosMenosUmEmCobra([H|T],POSICOESCOBRA,NLINHA):-
        validarTodosMenosUmEmCobraLinha(H,POSICOESCOBRA,NLINHA,1),
        NLINHA1 is NLINHA + 1,
        validarTodosMenosUmEmCobra(T,POSICOESCOBRA,NLINHA1).

verificarCobraConexa(TAB,TAMANHO,POSICOES):-
        reverse(POSICOES,POSICOES1),
        getListaPosicoesCobra(TAB,TAMANHO,POSICOES1,POSICOESCOBRA),
        validarTodosMenosUmEmCobra(TAB,POSICOESCOBRA,1).
```

Código fonte: random.pl

```
% Jogo Random

gerarTabuleiro([],_,[]).

gerarTabuleiro([_|T1],TAMANHO,[H|T]):-
        length(H,TAMANHO),
        gerarTabuleiro(T1,TAMANHO,T).

gerarTabuleiro(TAMANHO,TAB):-
        length(TAB1,TAMANHO),
        gerarTabuleiro(TAB1,TAMANHO,TAB).

inicializarTabuleiroRandomColuna([],[],_,_,_,_).

inicializarTabuleiroRandomColuna([_|T],[-
1|T1],NLINHA/NCOLUNA,LF/CF,NLINHA,NCOLUNA):-
        NCOLUNA1 is NCOLUNA + 1,
        inicializarTabuleiroRandomCo-
luna(T,T1,NLINHA/NCOLUNA,LF/CF,NLINHA,NCOLUNA1), !.

inicializarTabuleiroRandomColuna([_|T],[-
1|T1],LI/CI,NLINHA/NCOLUNA,NLINHA,NCOLUNA):-
        NCOLUNA1 is NCOLUNA + 1,
        inicializarTabuleiroRandomCo-
luna(T,T1,LI/CI,NLINHA/NCOLUNA,NLINHA,NCOLUNA1), !.

inicializarTabuleiroRandomColuna([_|T],[0|T1],LI/CI,LF/CF,NLINHA,NCOLUNA):-
        NCOLUNA1 is NCOLUNA + 1,
        inicializarTabuleiroRandomCo-
luna(T,T1,LI/CI,LF/CF,NLINHA,NCOLUNA1).

inicializarTabuleiroRandomLinha([],[],_,_,_).

inicializarTabuleiroRandomLinha([H|T],[H1|T1],LI/CI,LF/CF,NLINHA):-
        inicializarTabuleiroRandomColuna(H,H1,LI/CI,LF/CF,NLINHA,1),
        NLINHA1 is NLINHA + 1,
        inicializarTabuleiroRandomLinha(T,T1,LI/CI,LF/CF,NLINHA1).

inicializarTabuleiroRandom(TAB,LI/CI,LF/CF,TAB1):-
        inicializarTabuleiroRandomLinha(TAB,TAB1,LI/CI,LF/CF,1).
```