Starry Night

Programação em Logica

FEUP

Resolução de um problema de decisão

Grupo Starry Night 2:
Afonso Mendonça - up201706708
Filipe Nogueira - up201604129

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

Janeiro 5, 2020

Conteúdo

Resumo

O trabalho teve como objectivo a resolução de um problema de decisão, Starry Night, através da aplicação de restrições sobre domínios físicos e foi elaborado no âmbito da disciplina de Programação em Lógica inserida no 3º ano do Mestrado Integrado em Engenharia Informática e Computação.

1 Introdução

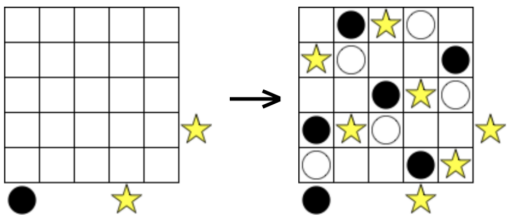
O presente trabalho centrou-se na aplicação dos conhecimentos adquiridos nas aulas teóricas e práticas de Programação em Lógica e visou a construção de um programa com capacidade para resolver um problema de decisão ou otimização em Prolog com restrições.

Com esse objectivo, foi implementada uma aplicação para encontrar soluções para o puzzle Starry Night. O puzzle tem estrutura em tabuleiro com igual número de linhas e colunas, e o seu preenchimento consiste na atribuição de '1' (estrela), '2' (lua) e '3' (sol).

O artigo enuncia o problema e apresenta a metodologia utilizada na obtenção das soluções. No artigo ainda são vertidas as funções responsáveis pela representação das soluções e as estatísticas relativas ao processo de obtenção da solução.

2 Descrição do problema

Starry Night é um puzzle criado por Erich Friedman. Este puzzle é constituído por uma grelha e três símbolos distintos: um círculo branco (sol), um círculo preto (lua) e uma estrela. Para atingir a solução de deste puzzle é necessário colocar um sol, uma lua e uma estrela em cada linha e coluna da grelha. Não é permitido símbolos idênticos se tocarem na diagonal. Um círculo ao lado da grelha representa a cor do círculo mais próximo à estrela. Uma estrela ao lado da grelha significa que ambos os círculos se encontram à mesma distância da estrela. Desta forma, Starry night é um problema de decisão. O algoritmo desenvolvido é capaz de encontrar uma solução para qualquer grelha criada de dimensão igual ou superior a 5x5.



3 Abordagem

Para encontrar uma solução para o problema exposto, recorremos à metodologia inerente à resolução de problemas de satisfação de restrições PSR. Foram definidas variáveis de decisão e respectivos domínios bem como as restrições que limitam os valores que as variáveis podem assumir.

As soluções encontradas correspondem à atribuição a cada variável de decisão dum valor único que respeite as restrições definidas.

No problema em análise às variáveis de decisão (células do tabuleiro) pode ser atribuído o símbolo sol, lua ou estrela. As restrições incluídas no programa ditam que em cada coluna e linha do tabuleiro existe um e só um símbolo sol, lua e estrela. Na célula diagonal adjacente a uma célula, não pode existir um símbolo igual ao símbolo que consta dessa célula. Segundo as restrições definidas pelo utilizador, este define, por linha e coluna, a posição relativa dos símbolos sol e lua em relação ao símbolo estrela como anteriormente explicitado.

3.1 Variáveis de Decisão

Considerando que a dimensão dos tabuleiros é NxN, o número de variáveis de decisão, correspondentes às células do tabuleiro, é igual a NxN. As variáveis de decisão podem assumir os valores 1, 2 e 3. Aos valores 1, 2 e 3 correspondem, respectivamente, os símbolos sol, lua e estrela.

Inicialmente, a todas as células do tabuleiro é atribuído o domínio de 0 (célula vazia). Na fase de labeling (geração da solução), é atribuído a algumas células, a solução 1, 2 ou 3, de acordo com as restrições impostas pelo utilizador.

3.2 Restrições

No âmbito no SICStus Prolog foram definidas como restrições rígidas as restrições descritas anteriormente quanto à obrigatoriedade de em cada linha e coluna existir um e só um símbolo sol, lua e estrela, bem como a impossibilidade de repetir na célula diagonal adjacente a cada célula, o símbolo que consta nessa célula.

No nosso problema aos símbolos sol, lua e estrela correspondem valores quantitativos de 1, 2 e 3 respectivamente. Os predicados desenvolvidos para garantir as restrições rígidas foram os seguintes:

3.2.1 check lines:

Neste predicado, é garantido que o somatório de cada linha e cada coluna é igual a 6. A cardinalidade de cada linha e coluna garante que os números 1, 2 e 3 figuram uma e só uma vez.

3.2.2 check row diagonals:

Neste predicado é garantido que as células diagonais, à esquerda e à direita de uma determinada célula, não podem assumir um valor igual ao atribuído a essa célula.

No âmbito do mesmo programa foram assumidas restrições flexíveis introduzidas pelo utilizador que, tal como referido anteriormente, se reportam à posição, por linha e coluna, dos símbolos sol e lua relativamente ao símbolo estrela.

3.2.3 enforce side restrictions:

Existem três variações deste predicado para o sol (1), lua (2) e estrela (3). É calculado um delta para os símbolos sol e lua que representam a distância relativa à estrela. No primeiro caso, o delta do sol tem que ser menor que o delta da lua. No segundo caso, o contrário, e no terceiro caso, estes últimos tem que ser iguais. Assim, neste último caso, garante-se a equidistância.

3.3 Estratégia de Pesquisa

Para encontrar a solução foi utilizada a estratégia de opção por omissão, ou seja, leftmost, step, up e satisfy:

labeling([], FTB) – [] é o mesmo que: [leftmost, step, up, satisfy].

3.3.1 Ordenação de Variáveis:

leftmost : variável mais à esquerda.

3.3.2 Seleção de Valores:

step: escolha binária entre X #= B e X (not)#= B, onde B é a lower ou upper bound de X.

3.3.3 Ordenação de Valores:

up: domínio explorado por ordem ascendente.

3.3.4 Soluções a Encontrar:

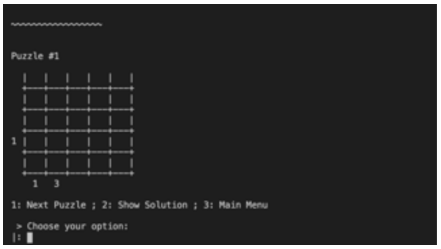
satisfy: todas as soluções são enumeradas por backtracking.

4 Visualização da Solução

Ao inicializar o programa, é necessário inserir "start." na consola. De seguida o utilizador irá deparar-se com o menu principal:

Deste ponto, o utilizador poderá escolher um dos 16 puzzles guardados (1), criar o seu próprio puzzle (2) ou sair do jogo (3).

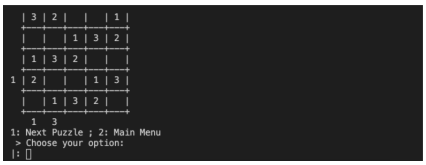
Ao selecionar a primeira opção, o utilizador é redirecionado para o ecrã abaixo apresentado, onde o predicado "show puzzle menu"é inicializado com o primeiro puzzle.



Aqui o utilizador pode escolher entre passar para o próximo puzzle (1), mostrar a solução do puzzle em questão (2) ou retornar para o menu principal. Adicionalmente, o puzzle é sempre apresentado em branco.

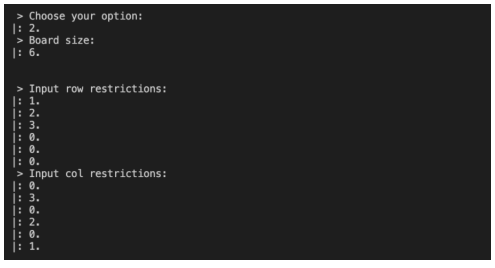
Ao selecionar a primeira opção do menu anterior, o utilizador depara-se com o ecrã anterior. O puzzle 2 é resultado do predicado "show puzzle menu", contudo o número do puzzle passado como argumento é 2.

Caso a segunda opção tenha sido escolhida, a solução do puzzle selecionado irá ser apresentada.



Neste caso, o predicado "show solved puzzle menu"é chamado com o PuzzleNo igual a 2, que por sua vez, por meio da função starrynight, o predicado "display solution"é inicializado. Este último é responsável pelo preenchimento do puzzle com a solução encontrada.

De volta ao menu principal, quando a segunda opção é selecionada, o utilizador consegue criar o seu próprio puzzle. O predicado "make your own menu"apresenta o seguinte menu:



Por motivos de demonstração, apenas se irá considerar um puzzle 6x6. As restrições deste novo tabuleiro são introduzidas no sentido última para primeira. Ou seja, a primeira restrição introduzida ('row restriction -¿ 1.') é a que vai ser aplicada no canto inferior esquerdo (primeira row). As colunas e as suas restrições se comportam de forma análoga.

Finalmente, a solução é mostrada da seguinte forma:

Esta solução é demonstrada utilizando, mais uma vez, o predicado "display solution".

5 Resultados

Tamanho do tabuleiro	Tempo total	Resumptions	Entailments	Prunings	Backtracks
5x5	1.912	4959	11359	19864	284
6x6	2.284	398669	122772	171157	2344
...					
10x10	2.644	21465	7674	9851	92

O problema não apresenta solução para tabuleiros de dimensão inferior a 5x5 devido às restrições impostas. Tabuleiros dessa dimensão ou não permitem a inclusão dos 3 símbolos numa linha / coluna (caso do tabuleiro 2x2), ou não permitem a não repetição do símbolo na diagonal adjacente. Para tabuleiros de dimensão superior a 10x10, a performance do código apresenta menor eficiência. A fiabilidade dos resultados apresentados na tabela anterior, encontra-se comprometida pela reduzida dimensão da amostra (4 puzzles para cada dimensão), dado que não foi implementado um gerador de puzzles.

6 Conclusões e Trabalho Futuro

O trabalho teve como objectivo implementar um programa com capacidade para atingir uma solução para qualquer variação do puzzle Starry Night, através da aplicação dos conceitos relativos à Programação em Lógica com restrições. O trabalho desenvolvido encontra solução para tabuleiros de dimensão igual ou superior a 5x5 e demonstra-se eficiente para tabuleiros de dimensões até 10x10. Para tabuleiros de dimensão superior a 10x10, a eficiência fica comprometida, dado que o processo para atingir uma solução possível torna-se muito lento. Referimos por último que, em toda a extensão deste artigo, quando mencionada a eficiência da aplicação, foi considerado que o utilizador impõe sempre um número razoável de restrições. Tendo em consideração o elevado número possibilidades de escolha, em termos de número de restrições flexíveis que o utilizador tem, a sua contabilização torna-se impraticável.

7 Bibiografia

Para a realização deste trabalho foram apenas utilizados os slides disponibilizados da cadeira Programação em Lógica.

8 Anexo

8.1 starynight.pl

```
:-use_module(library(clpfd)).
:-use_module(library(lists)).
:-consult('puzzles.pl').
:-consult('display.pl').
:-consult('menu.pl').
:-consult('utils.pl').

/*
    0: vazio
    1: Sol
    2: Lua
    3: Estrela
*/

/*
* Inicia o menu do programa.
*/
start:-
    menu.

/*
* Resolve um dos problemas pre-definidos, numerados de 1-16, especificado em Puzzle. A solução é dada em B.
* Começa por definir os domínios do tabuleiro, garantindo que cada linha tem apenas valores de 0-3. De seguida,
* garante-se que cada linha tem cada um único valor de cada possível (1/sol, 2/lua, 3/estrela) e que as linhas
* respeitam as restrições para as rows.A seguir, verifica as restrições diagonais, garantindo que símbolos
* iguais não se tocam diagonalmente. Por fim, faz as mesmas verificações que fez para as linhas mas numa
* matriz tranposta, ou seja, faz as verificações para as colunas. Depois, estando o labeling feito, faz display.
*/
starrynight(B, Puzzle):-
    puzzle(Puzzle, B, N, RestrictRows, RestrictCols),
    create_board_domains(B),
    check_lines(B, RestrictRows),
    enforce_diagonal_restrictions(B, B, N, 1),
    transpose(B, TB),
    check_lines(TB, RestrictCols),
    append(TB, FTB),
    %write('BEFORE LABELING\n\n'),
    labeling([], FTB),
    %write('AFTER LABELING\n\n'),
    write('  '), display_separator(N),
```

```
    display_solution(N, B, RestrictRows, RestrictCols).
    /*
    fd_statistics,
    statistics.
    */

/*
* Predicado com o mesmo propósito do anterior mas usado quando é o utilizador a fazer o seu próprio puzzle.
*/
starrynight(B, N, RestrictRows, RestrictCols):-
    create_board_domains(B),
    check_lines(B, RestrictRows),
    enforce_diagonal_restrictions(B, B, N, 1),
    transpose(B, TB),
    check_lines(TB, RestrictCols),
    append(TB, FTB),
    %write('BEFORE LABELING\n\n'),
    labeling([], FTB),
    %write('AFTER LABELING\n\n'),
    write('  '), display_separator(N),
    display_solution(N, B, RestrictRows, RestrictCols).
    /*
    fd_statistics,
    statistics.
    */

/*
* Menu para quando o puzzle feito pelo utilizador não tem qualquer solução. Os predicados anteriores
* apenas poderão falhar neste cenário, daí estar colocado aqui e não no menu.pl. Já os predicados
* para processar a escolha do user neste menu estão no menu.pl.
*/
starrynight(_, _, _, _):-
    write('No solution was found for your puzzle. Sorry! :('),nl,
    nl, write('1: Make Another ; 2: Main Menu'), nl, nl,
    write(' > Choose your option: '), nl,
    user_input(Option, 1, 2),
    process_make_your_own_input_option(Option).

/*
* Cria os domínios do tabuleiro, garantindo que os valores da matriz são apenas 0, 1, 2 ou 3
*/
create_board_domains([]).
create_board_domains([H|T]):-
    domain(H, 0, 3),
    create_board_domains(T).

/*
* Garante que apenas existe uma instância de cada figura na respetiva linha, sendo que zeros o número é indiferente
* pois são usados para representar um espaço vazio. Garante também que as restrições laterais são cumpridas.
* De seguida, segue recurssivamente para as seguintes linhas.
*/
check_lines([], []).
check_lines([H|T], [RH|RT]):-
    global_cardinality(H, [1-1, 2-1, 3-1, 0-_]),
    enforce_side_restriction(H, RH),
    check_lines(T, RT).

/*
* Os seguintes predicados forçam o cumprimento das restrições laterais dadas como argumento, podendo ser estas 0, 1, 2 ou 3
* No caso do 0, não há nenhuma. Quando é um 1, signifca que o sol(1) tem que estar mais perto da estrela(3) na respetiva li
* Quando é um 2, é o contrário do referido anteriormente. É calculada a posição dos respetivos elementos na linha e calcula
* a distância para a estrela. COnsoante a restrição, é garantida a maior/igual/menor proximidade das figuras na linha.
*/
enforce_side_restriction(_, 0).

enforce_side_restriction(L, 1):-
    element(SunPos, L, 1),
    element(MoonPos, L, 2),
    element(StarPos, L, 3),
    DeltaSun #= abs(StarPos - SunPos),
    DeltaMoon #= abs(StarPos - MoonPos),

    DeltaSun #< DeltaMoon.

enforce_side_restriction(L, 2):-
    element(SunPos, L, 1),
    element(MoonPos, L, 2),
    element(StarPos, L, 3),
    DeltaSun #= abs(StarPos - SunPos),
    DeltaMoon #= abs(StarPos - MoonPos),
```

```
    DeltaSun #> DeltaMoon.

enforce_side_restriction(L, 3):-
    element(SunPos, L, 1),
    element(MoonPos, L, 2),
    element(StarPos, L, 3),
    DeltaSun #= abs(StarPos - SunPos),
    DeltaMoon #= abs(StarPos - MoonPos),

    DeltaSun #= DeltaMoon.

/*
* Predicado recurssivo que, por sua vez, chama o check_row_diagonals para garantir o cumprimento
* das restrições diagonais. Isto é, que nenhuns dois símbolos iguais se podem tocar diagonalmente.
*/
enforce_diagonal_restrictions(_, _, NRows, NRows).
enforce_diagonal_restrictions([H|T], Board, NRows, CurrRow):-
    check_row_diagonals(H, Board, NRows, CurrRow, 1),
    NextRow is CurrRow + 1,
    enforce_diagonal_restrictions(T, Board, NRows, NextRow).

/*
* Para verificar que nenhumas duas células da matriz, diagonalmente vizinhas, têm o mesmo valor diferente
* de zero, ou seja, não têm a mesma figura, basta começar pela primeira linha e verificar apenas a de baixo.
* Para isto há três predicados diferentes para três cenários diferentes: quando se está a verificar a primeira
* célula da linha, a última célula da linha ou as restantes. Nos primeiros dois basta apenas verificar uma diagonal
* (esquerda e direita, respetivamente) enquanto que para o último cenário é preciso verificar a diagonal da esquerda
* e direita.
* Começa-se por calcular a linha inferior, sendo depois obtido as diagonais necessárias. Depois, caso o valor da célula
* atual seja 0, não é preciso nenhuma restrição. Caso não seja, aqui sim é preciso garantir que as diagonais são diferentes
* Estando estas restrições impostas, é confirmada o seguimento desta regra do jogo.
*/
check_row_diagonals([H|T], Board, NRows, CurrRowN, 1):-
    BelowRowN is CurrRowN + 1,
    RightCol is 2,
    nth1(BelowRowN, Board, BelowRow),
    element(RightCol, BelowRow, RightDiagonal),
    H#=0 #\/ H #\= RightDiagonal,
    %write('First Col of Row Finished'), nl,
    check_row_diagonals(T, Board, NRows, CurrRowN, RightCol).

check_row_diagonals([H|_], Board, NRows, CurrRowN, NRows):-
    CurrColN is NRows,
    BelowRowN is CurrRowN + 1,
    LeftCol is CurrColN - 1,
    nth1(BelowRowN, Board, BelowRow),
    element(LeftCol, BelowRow, LeftDiagonal),
    %write('Last Col of Row Finished'), nl,
    H#=0 #\/ H #\= LeftDiagonal.

check_row_diagonals([H|T], Board, NRows, CurrRowN, CurrColN):-
    BelowRowN is CurrRowN + 1,
    LeftCol is CurrColN - 1,
    RightCol is CurrColN + 1,
    nth1(BelowRowN, Board, BelowRow),
    element(LeftCol, BelowRow, LeftDiagonal),
    element(RightCol, BelowRow, RightDiagonal),
    H#=0 #\/ H #\= RightDiagonal,
    H#=0 #\/ H #\= LeftDiagonal,
    %write('Col of Row Finished'), nl,
    check_row_diagonals(T, Board, NRows, CurrRowN, RightCol).
```

8.2 menu.pl

```
/*
* Predicado que desenha o menu principal.
*/
display_menu:-
    nl,nl,
    write(' ____________________________________________________________________ '),nl,
    write('|                  *            *                                    |'),nl,
    write('|     *                          *         *       *         *      |'),nl,
    write('|        _____ _                                 _   _ _       _     _        |'),nl,
    write('|      / ____| |        *             | \\ | (_)  *  | |   | |    *   |'),nl,
    write('|     | (___ | |_ __ _ __ _ __ _ __   _   |  \\| |_  __ _| |__ | |_       |'),nl,
    write('|      \\___ \\| __/ _\` | \'__| \'__| | | | | . \` | |/ _\` | \'_ \\| __|      |'),nl,
    write('|      ____) | || (_| | |  | |  | |_| | | |\\  | | (_| | | | | |_       |'),nl,
    write('|     |_____/ \\__\\__,_|_|  |_|   \\__, | |_| \\_|_|\\__, |_| |_|\\__|      |'),nl,
    write('|  *                             __/ |           __/ |                |'),nl,
    write('|                    *          |___/     *     |___/    *            |'),nl,
    write('|                                                          *         |'),nl,
    write('|          *                                                         |'),nl,
    write('|                         1. Choose Puzzle      *          *         |'),nl,
    write('|                 *                                                  |'),nl,
    write('|      *                  2. Make Your Own                           |'),nl,
    write('|                                                             *      |'),nl,
    write('|                            3. Exit             *                   |'),nl,
    write('|                 *                                                  |'),nl,
    write('|    *                           *          *                 *     |'),nl,
    write('|____________________________________________________________________|'),nl,nl,nl.

/*
* Menu principal. Desenha a interface e pede input ao utilizador de forma a ele poder
* navegar para os diferentes menus.
*/
menu:-
    display_menu,
    write(' > Choose your option: '),
    nl,
    user_input(Option, 1, 3),
    process_menu_input(Option).

process_menu_input(1):-
    show_puzzle_menu(1).
```

```
process_menu_input(2):-
    make_your_own_menu.

process_menu_input(3).

/*
* Predicado de display do menu do puzzle #1. Tem um predicado exclusivo devido à opção de "Previous Puzzle",
* que não deve estar aqui presente.
*/
show_puzzle_menu(1):-
    nl,nl,nl,write('~~~~~~~~~~~~~~~~~~'),nl,nl,nl,
    write('Puzzle #'), write(1), nl, nl,
    puzzle(1, _B, N, RestrictRows, RestrictCols),
    ( N =:= 5 -> emptyMidBoard(EmptyB); emptyBigBoard(EmptyB) ),
    write('  '), display_separator(N), display_solution(N, EmptyB, RestrictRows, RestrictCols),
    nl, nl, write('1: Next Puzzle ; 2: Show Solution ; 3: Main Menu'), nl, nl,
    write(' > Choose your option: '), nl,
    user_input(Option, 1, 4),
    process_puzzle_menu_input(Option, 1).

/*
* Predicado de display do menu do puzzle #16. Tem um predicado exclusivo devido à opção de "Next Puzzle",
* que não deve estar aqui presente.
*/
show_puzzle_menu(16):-
    nl,nl,nl,write('~~~~~~~~~~~~~~~~~~'),nl,nl,nl,
    write('Puzzle #'), write(16), nl, nl,
    puzzle(16, _B, N, RestrictRows, RestrictCols),
    ( N =:= 5 -> emptyMidBoard(EmptyB); emptyBigBoard(EmptyB) ),
    write('  '), display_separator(N), display_solution(N, EmptyB, RestrictRows, RestrictCols),
    nl, nl, write('1: Previous Puzzle ; 2: Show Solution ; 3: Main Menu'), nl, nl,
    write(' > Choose your option: '), nl,
    user_input(Option, 1, 4),
    process_puzzle_menu_input(Option, 16).

/*
* Predicado de display do menu do puzzle dado em PuzzloeNo.
*/
show_puzzle_menu(PuzzleNo):-
    nl,nl,nl,write('~~~~~~~~~~~~~~~~~~'),nl,nl,nl,
```

```
    write('Puzzle #'), write(PuzzleNo), nl, nl,
    puzzle(PuzzleNo, _B, N, RestrictRows, RestrictCols),
    ( N =:= 5 -> emptyMidBoard(EmptyB); emptyBigBoard(EmptyB) ),
    write('  '), display_separator(N), display_solution(N, EmptyB, RestrictRows, RestrictCols),
    nl, nl, write('1: Next Puzzle ; 2: Previous Puzzle ; 3: Show Solution ; 4: Main Menu'), nl, nl,
    write(' > Choose your option: '), nl,
    user_input(Option, 1, 4),
    process_puzzle_menu_input(Option, PuzzleNo).

/*
* Predicado de display do menu do puzzle mas depois do user ter pedido a solução do mesmo,
* mostrando-o então resolvido.
*/
show_solved_puzzle_menu(PuzzleNo):-
    nl, nl, nl, write('Puzzle #'), write(PuzzleNo), write(' - Solution'), nl, nl,
    starrynight(_B, PuzzleNo),
    nl, nl, write('1: Next Puzzle ; 2: Main Menu'), nl, nl,
    write(' > Choose your option: '), nl,
    user_input(Option, 1, 2),
    process_solved_menu_input(Option, PuzzleNo).

/*
* Predicado de display do menu do puzzle mas depois do user ter pedido a solução do mesmo,
* mostrando-o então resolvido. Tem um predicado exclusivo devido à opção de "Next Puzzle",
* que não deve estar aqui presente.
*/
show_solved_puzzle_menu(16):-
    nl, nl, nl, write('Puzzle #'), write(16), write(' - Solution'), nl, nl,
    starrynight(_B, 16),
    nl, nl, write('1: Previous Puzzle ; 2: Main Menu'), nl, nl,
    write(' > Choose your option: '), nl,
    user_input(Option, 1, 2),
    process_solved_menu_input(Option, 16).

process_puzzle_menu_input(1, 1):-
    show_puzzle_menu(2).

process_puzzle_menu_input(2, 1):-
    show_solved_puzzle_menu(1).
```

```
process_puzzle_menu_input(3, 1):-
    menu.

process_puzzle_menu_input(1, 16):-
    show_puzzle_menu(15).

process_puzzle_menu_input(2, 16):-
    show_solved_puzzle_menu(16).

process_puzzle_menu_input(3, 16):-
    menu.

process_puzzle_menu_input(1, PuzzleNo):-
    PuzzleNo1 is PuzzleNo + 1,
    show_puzzle_menu(PuzzleNo1).

process_puzzle_menu_input(2, PuzzleNo):-
    PuzzleNo1 is PuzzleNo - 1,
    show_puzzle_menu(PuzzleNo1).

process_puzzle_menu_input(3, PuzzleNo):-
    show_solved_puzzle_menu(PuzzleNo).

process_puzzle_menu_input(4, _PuzzleNo):-
    menu.

process_solved_menu_input(1, 16):-
    show_puzzle_menu(15).

process_solved_menu_input(1, PuzzleNo):-
    PuzzleNo1 is PuzzleNo + 1,
    show_puzzle_menu(PuzzleNo1).

process_solved_menu_input(2, _PuzzleNo):-
    menu.

/*
* Menu de criação de puzzles pelo user. Aqui ele pode escolher o tamanho do tabuleiro e as restrições
* para as colunas e linhas. Limite de 5-15 no tamanho do tabuleiro pois, para valores inferior a 5, os possíveis puzzles
* são pouquíssimos. Enquanto que para valores acima de 15 o tempo de cálculo é demasiado grande.
```

```
*/
make_your_own_menu:-
    nl,nl,nl,write('~~~~~~~~~~~~~~~~~'),nl,nl,nl,
    write('Make your own puzzle!'),
    nl, nl,write(' > Board size (min: 5, max: 15): '), nl,
    user_input(Size, 5, 15),
    nl,
    write(' > Input row restrictions: '), nl,
    get_n_inputs(0, 3, Size, RowRestrictions),
    nl, nl, write(' > Input col restrictions: '), nl,
    get_n_inputs(0, 3, Size, ColRestrictions),
    nl, nl, write('Your puzzle:'), nl, nl,
    buildBoard(Size, B),
    starrynight(B, Size, RowRestrictions, ColRestrictions),
    nl, nl, write('1: Make Another ; 2: Main Menu'), nl, nl,
    write(' > Choose your option: '), nl,
    user_input(Option, 1, 2),
    process_make_your_own_input_option(Option).

/*
* Processar input do utilizador no respetivo menu.
*/
process_make_your_own_input_option(1):-
    make_your_own_menu.

/*
* Processar input do utilizador no respetivo menu.
*/
process_make_your_own_input_option(2):-
    menu.
```

8.3 puzzles.pl

```
/*
* Ficheiro com tabuleiros não instanciados de tamanho pré-definido assim como os puzzles exemplo do criador do jogo.
* Puzzles obtidos de: https://www2.stetson.edu/~efriedma/puzzle/night/
*/

midBoard([
    [_, _, _, _, _],
    [_, _, _, _, _],
    [_, _, _, _, _],
    [_, _, _, _, _],
    [_, _, _, _, _]
    ], 5).

bigBoard([
    [_, _, _, _, _, _],
    [_, _, _, _, _, _],
    [_, _, _, _, _, _],
    [_, _, _, _, _, _],
    [_, _, _, _, _, _],
    [_, _, _, _, _, _]
    ], 6).

puzzle(1, Board, Size, RowRestrictions, ColRestrictions):-
    midBoard(Board, Size),
    append([], [0, 0, 0, 1, 0], RowRestrictions),
    append([], [1, 3, 0, 0, 0], ColRestrictions).

puzzle(2, Board, Size, RowRestrictions, ColRestrictions):-
    midBoard(Board, Size),
    append([], [0, 0, 0, 2, 0], RowRestrictions),
    append([], [0, 3, 3, 0, 0], ColRestrictions).

puzzle(3, Board, Size, RowRestrictions, ColRestrictions):-
    midBoard(Board, Size),
    append([], [0, 0, 0, 3, 0], RowRestrictions),
    append([], [0, 3, 2, 0, 0], ColRestrictions).

puzzle(4, Board, Size, RowRestrictions, ColRestrictions):-
    midBoard(Board, Size),
    append([], [0, 0, 0, 1, 0], RowRestrictions),
    append([], [0, 3, 2, 0, 0], ColRestrictions).

puzzle(5, Board, Size, RowRestrictions, ColRestrictions):-
    bigBoard(Board, Size),
    append([], [3, 0, 3, 0, 0, 0], RowRestrictions),
    append([], [0, 3, 3, 0, 2, 0], ColRestrictions).

puzzle(6, Board, Size, RowRestrictions, ColRestrictions):-
    bigBoard(Board, Size),
    append([], [0, 0, 3, 0, 3, 3], RowRestrictions),
    append([], [1, 0, 0, 0, 3, 0], ColRestrictions).

puzzle(7, Board, Size, RowRestrictions, ColRestrictions):-
    bigBoard(Board, Size),
    append([], [0, 3, 2, 3, 0, 0], RowRestrictions),
    append([], [0, 0, 3, 1, 1, 0], ColRestrictions).

puzzle(8, Board, Size, RowRestrictions, ColRestrictions):-
    bigBoard(Board, Size),
    append([], [3, 0, 0, 0, 1, 0], RowRestrictions),
    append([], [0, 2, 1, 0, 3, 3], ColRestrictions).

puzzle(9, Board, Size, RowRestrictions, ColRestrictions):-
    bigBoard(Board, Size),
    append([], [1, 1, 2, 0, 0, 3], RowRestrictions),
    append([], [0, 0, 1, 0, 1, 3], ColRestrictions).

puzzle(10, Board, Size, RowRestrictions, ColRestrictions):-
    bigBoard(Board, Size),
    append([], [0, 0, 2, 0, 2, 1], RowRestrictions),
    append([], [3, 3, 1, 0, 0, 0], ColRestrictions).

puzzle(11, Board, Size, RowRestrictions, ColRestrictions):-
    bigBoard(Board, Size),
    append([], [0, 1, 1, 1, 0, 0], RowRestrictions),
    append([], [3, 1, 0, 0, 3, 1], ColRestrictions).

puzzle(12, Board, Size, RowRestrictions, ColRestrictions):-
    bigBoard(Board, Size),
    append([], [1, 3, 2, 0, 2, 0], RowRestrictions),
```

```
    append([], [2, 3, 1, 0, 0, 0], ColRestrictions).

puzzle(13, Board, Size, RowRestrictions, ColRestrictions):-
    bigBoard(Board, Size),
    append([], [2, 1, 2, 1, 2, 0], RowRestrictions),
    append([], [0, 2, 0, 3, 1, 1], ColRestrictions).

puzzle(14, Board, Size, RowRestrictions, ColRestrictions):-
    bigBoard(Board, Size),
    append([], [3, 1, 0, 2, 2, 0], RowRestrictions),
    append([], [2, 2, 1, 2, 0, 2], ColRestrictions).

puzzle(15, Board, Size, RowRestrictions, ColRestrictions):-
    bigBoard(Board, Size),
    append([], [1, 1, 2, 1, 1, 1], RowRestrictions),
    append([], [2, 1, 1, 2, 1, 1], ColRestrictions).

puzzle(16, Board, Size, RowRestrictions, ColRestrictions):-
    bigBoard(Board, Size),
    append([], [1, 2, 1, 2, 2, 1], RowRestrictions),
    append([], [2, 2, 1, 2, 1, 2], ColRestrictions).
```

8.4 display.pl

```
/*
* Predicados de vizualização de tabuleiros.
*/

display_solution(_N, [], [], RC):-
    write('\n  '),
    display_col_restrictions(RC).

display_solution(N, [R|Rs], [RH|RT], RC):-
    nl,
    display_cell(RH),
    write(' | '),
    display_row(R, 0),
    display_solution(N, Rs, RT, RC).

display_row([], N):-
    write('\n  '),
    display_separator(N).
display_row([H|T], N):-
    N1 is N + 1,
    display_cell(H),
    write(' | '),
    display_row(T, N1).

display_separator(0):-
    write('+').
display_separator(N):-
    write('+---'),
    N1 is N - 1,
    display_separator(N1).

display_col_restrictions([]).
display_col_restrictions([H|T]):-
    write('  '),
    display_cell(H),
    write(' '),
    display_col_restrictions(T).


/*
* Caso não se esteja a utilizar Sicstus no Windows, pode-se descomentar as linhas abaixo para utilizar
* as respetivas figuras do Sol, Lua e Estrela em vez das suas representações numéricas (1, 2 e 3).
*/
display_cell(0):- write(' ').
%display_cell(1):- put_code(9675).
%display_cell(2):- put_code(9679).
%display_cell(3):- put_code(128970).
display_cell(1):- write('1').
display_cell(2):- write('2').
display_cell(3):- write('3').
```

8.5 utils.pl

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

/*
* Ficheiro com predicados para diversos usos.
*/

/*
* Obtenção de input do utilizador. Usado para navegar nos menus e para o utilizador construir o seu próprio puzzle.
*/
user_input(Input, Min, Max):-
    catch(read(Input),_Err,fail),
    integer(Input),
    between(Min, Max, Input).

user_input(Input, Min, Max):-
    nl,
    write('Invalid Input. Retry:'),
    nl,
    nl,
    user_input(Input, Min, Max).

/*
* Obtenção n inputs do utilizador entre Min e Max.
*/
get_n_inputs(Min, Max, N, R):-
    get_n_inputs_aux(Min, Max, N, [], RR),
    reverse(RR, R).

get_n_inputs_aux(_, _, 0, RF, RF).
get_n_inputs_aux(Min, Max, N, R, RF):-
    user_input(Input, Min, Max),
    N1 is N - 1,
    get_n_inputs_aux(Min, Max, N1, [Input|R], RF).

/*
* Predicado para "construir" tabuleiros com valores não instanciados de tamanho personalizado.
*/
buildBoard(N, B) :-
    length(B, N),
    map_list(length_list(N), B).
```

```
map_list(_, []).
map_list(C, [X|Xs]) :-
    call(C,X),
    map_list(C, Xs).

length_list(N, L) :-
    length(L, N).

/*
* Tabuleiros vazios de tamanho pré-definido usados na visualização dos puzzles pré-definidos.
*/
emptyMidBoard(
    [[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]]
    ).

emptyBigBoard(
    [[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]]
    ).
```