Shapely Squares

Resolução de Problema de Decisão usando Programação em Lógica com Restrições

Ângelo Daniel Pereira Mendes Moura

up201303828@fe.up.pt

César Alexandre da Costa Pinho

up201604039@fe.up.pt

FEUP-PLOG, Turma 3MIEIC05, Grupo Shapely Squares_3

Resumo. Neste artigo descreve-se um projeto realizado no âmbito da unidade curricular de Programação em Lógica, do Mestrado Integrado em Engenharia Informática e de Computação. O projeto consiste num programa escrito na linguagem de Programação em Lógica e desenvolvido no SICStus Prolog capaz de calcular a solução a de puzzles do tipo Shapely Squares, que exemplificam um problema de decisão combinatória.

Keywords.: Shapely Squares, Sicstus, Prolog, FEUP.

Instruções: Para testar o programa use o Sicstus para consultar o ficheiro shapelySquares.pl e corra o predicado shapely_square(N). N é um valor entre 1 e 13 que seleciona o puzzle a resolver do ficheiro puzzles.pl.

1 Introdução

Neste artigo descreve-se um projeto realizado no âmbito da unidade curricular de Programação em Lógica do 3º ano do curso Mestrado Integrado em Engenharia Informática e de Computação. O projeto proposto envolvia o estudo de um problema de decisão ou otimização e o desenvolvimento de um programa em Prolog capaz de calcular a sua solução. O grupo escolheu estudar um problema de decisão na forma de um puzzle denominado Shapely Squares.

Este artigo descreve detalhadamente o puzzle Shapely Squares, a sua decomposição em predicados simples e programáveis em Prolog e a maneira como a solução encontrada é representada. O artigo contém também estáticas retiradas de um estudo realizado sobre a solução desenvolvida assim como as conclusões que se podem observar a partir das mesmas.

2 Descrição do Problema

O puzzle Shapely Squares consiste num tabuleiro rectangular dividido em linhas e colunas formando assim um padrão em grelha constituído por células. À direita de cada linha é apresentado a soma total dos dígitos nessa linha e algumas das células contêm uma figura representativa de uma restrição adicional imposta nessa célula.

Para resolver o puzzle cada célula deve ser preenchida com um número de 0 a 9 de maneira a que a soma dos valores de cada linha seja igual aos fornecidos e os dígitos que se encontrem em células com uma figura respeitem a restrição por esta imposta.

Existem 7 figuras diferentes e portanto 7 restrições diferentes:

1. **A estrela)** Esta figura obriga o dígito da célula a ser primo, pelo menos 2 e a não ter vizinhos ortogonais que sejam primos ou 1.
2. **O quadrado)** Esta figura obriga o dígito da célula a ser ou 0 ou 5, mas o digito não pode ter o mesmo valor que um dos seus vizinhos, a não ser que esse vizinho seja uma célula com um diamante.
3. **O diamante)** Esta figura obriga o dígito a ser impar e a ser a soma de todos os dígitos que se encontram á sua esquerda na linha.
4. **O triângulo)** Esta figura obriga o dígito a estar posicionado diretamente debaixo de um dígito par e a ser inferior a este, mas não pode ser zero.
5. **O círculo)** Esta figura obriga o dígito a não ser um múltiplo de 3, e todos os dígitos em células com círculos devem ter o mesmo valor.
6. **O cavalo)** Esta figura obriga que o dígito nesta célula seja igual ao número de dígitos pares nas células acessíveis segundo o movimento de um cavalo de xadrez.
7. **O coração)** Esta figura obriga que os corações vizinhos devem somar um total de 10.

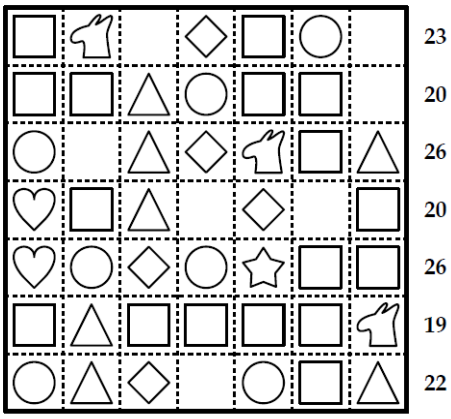

Fig. 1. Exemplo de um tabuleiro de Shapely Squares com dimensões 7X7.

3 Abordagem

Usando a linguagem de Prolog, o tabuleiro é representado como uma lista de listas, onde cada elemento representa uma célula do tabuleiro. Acrescentando ao tabuleiro uma lista contendo as somas o puzzle é totalmente representado. (Esta estrutura tabuleiro mais somas será referida de seguida como puzzle.)

Inicialmente o puzzle tem as células do seu tabuleiro preenchidas com valores de 0 a 7. Sendo que estes valores representam as figuras presentes nessa posição do tabuleiro de acordo com a seguinte legenda:

0-Célula sem figura.
1-Estrela.
2-Quadrado.
3-Diamante.
4-Circulo.
5-Triangulo.
6-Cavalo.
7-Coração.

No puzzle que representa a solução cada célula do tabuleiro terá um valor de 0 a 9 que corresponde á solução do puzzle.

```
puzzle(1, [ [2,6,0,3,2,4,0],
            [2,2,5,4,2,2,0],
            [4,0,5,3,6,2,5],
            [7,2,5,0,3,0,2],
            [7,4,3,4,1,2,2],
            [2,5,2,2,2,2,6],
            [4,5,3,0,4,2,5]],
            [23,20,26,20,26,19,22]).
```

Fig. 2. Representação do tabuleiro da Fig1 em Prolog

3.1 Variáveis de Decisão

A solução do puzzle é encontrada ao preencher cada célula com o valor correto. Este valor é determinado em consequência das restrições colocadas pelas figuras e pela soma dos valores de cada linha. No entanto, inicialmente é considerado que cada célula pode ter qualquer valor entre 0 e 9. Ou seja, cada célula do tabuleiro tem D={0,1,2,3,4,5,6,7,8,9}.

3.2 Restrições

O puzzle Shapely Squares tem no total 8 restrições gerais. Uma restrição para cada uma das 7 figuras e a soma de cada linha que também tem que ser respeitada.

Cada uma das restrições impingidas pelas figuras pode ser desconstruída em restrições mais concretas e programáveis em Prolog. Nesta secção apenas o cabeçalho destes predicados é apresentado, mas estes encontram-se na sua totalidade na secção de Anexos.

Nesta fase do artigo, usa-se a nomenclatura Sd para representar um subconjunto do domínio.

Note-se que o predicado apply_contraint_cell\5 usa a legenda fornecida em cima.

Estrela).

- Restringe o domínio do dígito da célula respetiva para o subconjunto Sd={2,3,5,7}.
- Restringe o domínio dos dígitos das células ortogonalmente vizinhas para Sd={0,4,6,8,9}.

O predicado em Prolog que representa a restrição da estrela é o seguinte:

```
apply_constraint_cell(1,SolvedPuzzle,_,X,Y):-
```

Este predicado usa o predicado star_constraint\3 para restringir o domínio das células vizinhas. Este predicado tem a seguinte forma:

```
star_constraint(SolvedPuzzle,X,Y):-
```

Quadrado).

- Restringe o domínio do dígito da célula respetiva para o subconjunto Sd={0,5}.
- O dígito tem que ter um valor diferente dos seus vizinhos que não tem a figura diamante.

O predicado em Prolog que representa a restrição do quadrado é o seguinte:

```
apply_constraint_cell(2,SolvedPuzzle,Puzzle,X,Y):-
```

Este predicado usa o predicado square_constraint\5 para verificar que o dígito não tem o mesmo valor que os seus vizinhos não diamantes. Este predicado tem a seguinte forma:

```
square_constraint(SolvedPuzzle,Puzzle,X,Y,V):-
```

Diamante).

- O dígito nesta célula não é divisível por 2.
- O dígito nesta célula é igual a soma dos dígitos que se encontram á sua esquerda na linha.

O predicado em Prolog que representa a restrição do diamante é o seguinte:

```
apply_constraint_cell(3,SolvedPuzzle,_,X,Y):-
```

Este predicado usa o predicado prefix_length\3 para formar uma lista com os dígitos nas células à esquerda do diamante para calcular a sua soma.
Triângulo).

- O dígito da célula diretamente em cima é divisível por 2.
- O dígito nesta célula é inferior ao dígito diretamente em cima.
- O dígito nesta célula é diferente de 0.
- O dígito da célula diretamente em cima é 2 ou superior a 2 (consequência lógica da restrição 3).

Importante notar que devido às restrições do triângulo referirem um dígito diretamente em cima, nenhum tabuleiro valido de Shapely Squares contem um triângulo na primeira linha.

O predicado em Prolog que representa a restrição do triângulo é o seguinte:

```
apply_constraint_cell(5,SolvedPuzzle,_,X,Y):-
```

Círculo).

- O dígito nesta célula não é divisível por 3.
- Todos os dígitos em células com círculos têm o mesmo valor.

As restrições impostas pelo círculo são garantidas usando dois predicados, ambos com o cabeçalho circle_constraint\3. O primeiro garante que o dígito é divisível por 3 e o segundo garante que todos os círculos contêm o mesmo dígito.

```
circle_constraint(SolvedPuzzle,X,Y):-
```

Estes predicados são invocados pelo predicado seguinte:

```
apply_constraint_cell(4,SolvedPuzzle,_,X,Y):-
```

Cavalo).

- O dígito nesta célula é igual ao número de dígitos divisíveis por 2 que se pode aceder seguindo o movimento de um cavalo de xadrez.

Como já foi referido esta figura utiliza o movimento do cavalo de xadrez na sua lógica, devido a isso foi necessário programar uma representação destes movimentos em Prolog. Para isso usa-se o seguinte predicado:

```
knight_attack_range\3.
```

Usando esta representação, o predicado seguinte pode facilmente aceder às células relevantes e verificar se os dígitos nessas células são divisíveis por 2.

```
knight_constraint(SolvedPuzzle,X,Y,AttackId,Count):-
```

Depois é só necessário restringir o dígito da célula com o cavalo para ser igual ao número de dígitos divisíveis por 2. Isto é feito no predicado seguinte:

```
apply_constraint_cell(6,SolvedPuzzle,_,X,Y):-
```

Coração).

- A soma do dígito desta célula com os dígitos de células vizinhas que também contêm um coração é 10.

O predicado em Prolog que representa a restrição do coração é o seguinte:

```
apply_constraint_cell(7,SolvedPuzzle,Puzzle,X,Y):-
```

Este predicado usa o predicado heart_constraint\5 para ir buscar os dígitos de células vizinhas que também têm um coração. Este predicado tem a seguinte forma:

```
heart_constraint(SolvedPuzzle,Puzzle,X,Y,V):-
```

4 Visualização da Solução

A apresentação da solução é bastante simples. São usados alguns predicados para garantir uma fácil compreensão do resultado, mas como a solução é uma lista de listas preenchida com valores numéricos a representação da mesma não contem dificuldade nenhuma.

Importante notar que a estrutura que contem a solução não contem as figuras, as figuras são mantidas na estrutura inicial e a solução é uma lista de lista do mesmo tamanho que o tabuleiro inicial. Devido a isto ambos os tabuleiros são imprimidos pelo programa.

Os predicados usados para imprimir a solução são display_game\3, display_legend\0, display_lin_separ\1, display_lin_separ\2, display_puzzle\4, display_line\2 e write_sum\1.

```
S - Star
Q - Square
D - Diamond
C - Circle
T - Triangle
K - Knight
H - Heart

-------------------------  -------------------------
| Q | K |   | D | Q | C |   | 23 |  | 0 | 3 | 6 | 9 | 0 | 4 | 1 | 23 |
-------------------------  -------------------------
| Q | Q | T | C | Q | Q |   | 20 |  | 5 | 0 | 4 | 4 | 5 | 0 | 2 | 20 |
-------------------------  -------------------------
| C |   | T | D | K | Q | T | 26 |  | 4 | 1 | 2 | 7 | 6 | 5 | 1 | 26 |
-------------------------  -------------------------
| H | Q | T |   | D |   | Q | 20 |  | 7 | 0 | 1 | 1 | 9 | 2 | 0 | 20 |
-------------------------  -------------------------
| H | C | D | C | S | Q | Q | 26 |  | 3 | 4 | 7 | 4 | 3 | 0 | 5 | 26 |
-------------------------  -------------------------
| Q | T | Q | Q | Q | Q | K | 19 |  | 5 | 2 | 0 | 5 | 0 | 5 | 2 | 19 |
-------------------------  -------------------------
| C | T | D |   | C | Q | T | 22 |  | 4 | 1 | 5 | 7 | 4 | 0 | 1 | 22 |
-------------------------  -------------------------
```

Fig. 3. Solução ao tabuleiro da Fig.1 apresentado pelo programa.

5 Resultados

De seguida apresenta-se uma tabela com os resultados da avaliação do nosso programa perante a resolução de diferentes puzzles.

Note-se que devido à falta da geração automática de novos puzzles a recolha de dados foi limitada a apenas 13 puzzles diferentes.

ID	Nº Figuras Diferentes	Dimensões	Retomas	Cortes(Prunings)	Backtracks	Restrições	Tempo
1	7	7x7	1687	1072	116	588	63
2	7	7x7	1789	1150	135	632	63
3	7	7x7	1637	1012	142	606	63
4	5	4x5	356	240	47	157	63
5	5	4x5	359	252	52	172	63
6	5	4x5	475	318	49	187	63
7	5	4x5	435	312	54	155	62
8	5	7x7	946	667	115	401	62
9	5	7x7	1205	797	114	484	62
10	4	4x4	301	205	40	132	63
11	4	4x4	295	253	39	153	63
12	4	4x4	300	200	39	140	63
13	4	4x4	727	519	48	171	62

Fig. 4. Tabela dos valores obtidos do estudo do programa perante diferentes puzzles.

Analisando a tabela em cima, os dados parecem informar que o tempo de resolução é independente do tamanho do tabuleiro. Na realidade isso não é verdade e quando maior o tabuleiro mais tempo o programa irá levar a encontrar a solução. Neste exemplo os tabuleiros usados não são muito diferentes em termos de dimensões para observar esse fenómeno.

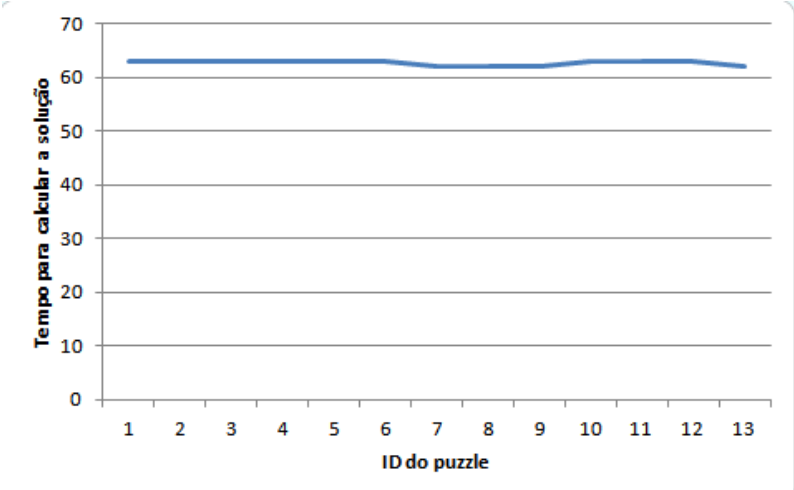

Fig. 5. Gráfico a comparar o tempo de resolução para cada puzzle.

No entanto já é possível observar o aumento do número de backtrakings quanto maior o tabuleiro for. E como sabemos que quanto maior o número de backtrakings maior o tempo de execução, é nestes valores que é possível ver o afeto que o tamanho do tabuleiro tem no tempo.

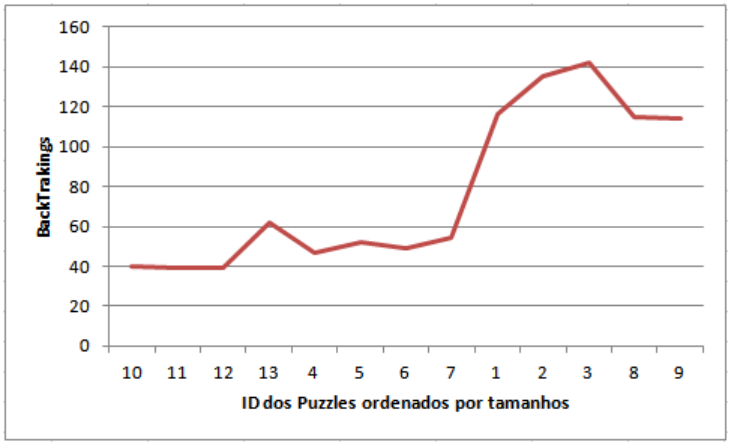

Fig. 6. Gráfico que demonstra o aumento dos Bactrakings com o aumento das dimensões do tabuleiro.

6 Conclusões

O projeto demonstra a utilidade e facilidade de Prolog para resolver problemas de decisão. A velocidade e eficiência da solução é bastante surpreendente, no entanto também ficou claro para nós que esta linguagem é uma ferramenta extremamente especializada que talvez deva ser usada em complemento com outras mais comodas.

Note-se que o projeto podia ser melhorado, com mais tempo disponível e mais pesquisa é possível que arranjássemos um método mais eficiente e apropriado para calcular a solução. Também não foi possível, devido a restrições de tempo acabar a geração aleatória de novos puzzles.

Em suma, o projeto foi concluído com sucesso, o programa calcula a solução correta de cada puzzle, e o seu desenvolvimento contribuiu positivamente para uma melhor compreensão do funcionamento do labeling e variáveis de decisão, assim como na aplicação de restrições. Como mais algum tempo, o grupo sente-se plenamente capaz de o refinar.

7 Bibliografia

1. Regras do Shapely Squares, https://thegriddle.net/puzzledir/shapelysquares_2010_06_15.pdf
2. SICStus Prolog, https://sicstus.sics.se/
3. SWI-Prolog, http://www.swi-prolog.org/

8 Anexo

Código Fonte.

shapelySquares.pl

```
%%%%% TYPE CODING %%%%%
% empty     :: 0 ::   %
% star      :: 1 :: S %
% square    :: 2 :: Q %
% diamond   :: 3 :: D %
% circle    :: 4 :: C %
% triangle  :: 5 :: T %
% knight    :: 6 :: K %
% heart     :: 7 :: H %
%%%%%%%%%%%%%%%%%%%%%%

:- include('Utilitarios.pl').
:- include('puzzles.pl').
:- use_module(library(clpfd)).
:- use_module(library(lists)).

display_header :-
    wri-
te('**************************************************
**\n'),
    write('****
****\n'),
    write('****                 Shapely Squares
****\n'),
    write('****
****\n'),
    wri-
te('**************************************************
**\n').

display_start_menu :-
    display_header,
    write('****
****\n'),
    write('****                    1 - Solucao
****\n'),
    write('****                    2 - Geracao
****\n'),
```

```
    write('****                    3 - About
****\n'),
    write('****                    4 - Sair
****\n'),
    write('****
****\n'),
    wri-
te('**************************************************
**\n').

start :-
    NumberOfPuzzle is 1,
    clr,
    display_start_menu,
    get_code(Code),
    skip_line,
    Choice is Code - 48,
    (
        Choice =:= 1 -> shapely_square(NumberOfPuzzle);
        Choice =:= 2 -> generate_menu;
        Choice =:= 3 -> about_menu;
        Choice =:= 4;
        start
    ).

generate_menu :-
    write('Digite um valor entre 2 e 9 (inclusive)\n'),
    write(' Columns :'),
    get_code(Code),
    skip_line,
    XSize is Code - 48,
    write(' Lines :'),
    get_code(Code),
    skip_line,
    YSize is Code - 48,
    generate(XSize,YSize).

display_about_menu :-
    display_header,
    write('****
****\n'),
    write('****                    Autores:
****\n'),
    write('****                - Angelo Moura
****\n'),
```

```
    write('****                - Cesar Pinho
****\n'),
    write('****
****\n'),
    write('****                   4 - Voltar
****\n'),
    write('****
****\n'),
    wri-
te('**************************************************
**\n').

about_menu :-
    clr,
    display_about_menu,
    get_code(Code),
    skip_line,
    Choice is Code - 48,
    (
        Choice =:= 4 -> start;
        about_menu
    ).

shapely_square(PuzzleNo) :-
    statistics(walltime,[Start|_]),

    abolish(circle_value/2),abolish(puzzle_size/2),
    puzzle(PuzzleNo,Puzzle,LineSums),
    length(Puzzle,YSize),
    nth1(1,Puzzle,Line),
    length(Line,XSize),
    asserta(puzzle_size(XSize,YSize)),
    new_puzzle(LineSums,SolvedPuzzle),

    asserta(circle_value(0,0)),
    apply_constraints(SolvedPuzzle,Puzzle,1,1),
    append(SolvedPuzzle,List),

    statistics(walltime,[BeforeLabeling|_]),
    labeling([],List),

    fd_statistics,
    statistics(walltime,[Stop|_]),

    ConstraintRuntime is BeforeLabeling - Start,
```

```
    LabelingRuntime is Stop - BeforeLabeling,
    TotalRuntime is Stop - Start,
    format('Runtime :\n  Before labeling : ~d\n  Labeling
: ~d\n  Total :
~d\n',[ConstraintRuntime,LabelingRuntime,TotalRuntime]),
    display_game(Puzzle, SolvedPuzzle, LineSums).


new_puzzle([],[]).
new_puzzle([Sum1|Sums], Puzzle):-
    new_line(9,Sum1,Line),
    new_puzzle(Sums,P1),
    append([Line],P1,Puzzle).

new_line(Limit,Sum,Line):-
    puzzle_size(X,_),
    length(Line,X),
    domain(Line,0,Limit),
    sum(Line,#=,Sum).

%%%%% Constraint apply
apply_constraints(_,_,_,Y):-
    puzzle_size(_,YSize),
    Y #> YSize .

apply_constraints(SolvedPuzzle,Puzzle,X,Y):-
    puzzle_size(XSize,_),
    X #> XSize,
    NewY #= Y + 1,
    apply_constraints(SolvedPuzzle,Puzzle,1,NewY).

apply_constraints(SolvedPuzzle,Puzzle,X,Y):-
    nth1(Y,Puzzle,Line),
    nth1(X,Line,Type),
    apply_constraint_cell(Type,SolvedPuzzle,Puzzle,X,Y),
    NewX #= X + 1,
    apply_constraints(SolvedPuzzle,Puzzle,NewX,Y).

%% Apply constraints to a cell
apply_constraint_cell(1,SolvedPuzzle,_,X,Y):-
    nth1(Y,SolvedPuzzle,Line),
    element(X, Line,V1), V1 in {2,3,5,7},
    PreviousY #= Y - 1,
    NextY #= Y + 1,
    PreviousX #= X - 1,
```

```
    NextX #= X + 1,
    star_constraint(SolvedPuzzle,X,PreviousY),
    star_constraint(SolvedPuzzle,PreviousX,Y),
    star_constraint(SolvedPuzzle,NextX,Y),
    star_constraint(SolvedPuzzle,X,NextY).

apply_constraint_cell(2,SolvedPuzzle,Puzzle,X,Y):-
    nth1(Y,SolvedPuzzle,Line),
    element(X,Line,V), V in {0,5},
    PreviousY #= Y - 1,
    NextY #= Y + 1,
    PreviousX #= X - 1,
    NextX #= X + 1,
    square_constraint(SolvedPuzzle,Puzzle,X,PreviousY,V),
    square_constraint(SolvedPuzzle,Puzzle,PreviousX,Y,V),
    square_constraint(SolvedPuzzle,Puzzle,NextX,Y,V),
    square_constraint(SolvedPuzzle,Puzzle,X,NextY,V).

apply_constraint_cell(3,SolvedPuzzle,_,X,Y):-
    nth1(Y,SolvedPuzzle,Line),
    element(X,Line,V),
    LeftSize #= X - 1,
    prefix_length(Line,LeftDigits,LeftSize),
    sum(LeftDigits,#=,V),
    V mod 2 #= 1.

apply_constraint_cell(4,SolvedPuzzle,_,X,Y):-
    circle_constraint(SolvedPuzzle,X,Y).

apply_constraint_cell(5,SolvedPuzzle,_,X,Y):-
    PreviousY #= Y - 1,
    !, % Tem que existir uma linha em cima de um triangu-
lo.
    nth1(PreviousY,SolvedPuzzle,PreviousLine),
    element(X,PreviousLine,AboveV),
    nth1(Y,SolvedPuzzle,Line),
    element(X,Line,V),
    AboveV #>= 2,
    AboveV mod 2 #= 0,
    V #> 0,
    V #< AboveV.

apply_constraint_cell(6,SolvedPuzzle,_,X,Y):-
    knight_constraint(SolvedPuzzle,X,Y,1,EvenCount),
    nth1(Y,SolvedPuzzle,Line),
```

```
    element(X,Line,V),
    V #= EvenCount.

apply_constraint_cell(7,SolvedPuzzle,Puzzle,X,Y):-
    nth1(Y,SolvedPuzzle,Line),
    element(X, Line,V),
    PreviousY #= Y - 1,
    NextY #= Y + 1,
    PreviousX #= X - 1,
    NextX #= X + 1,
    heart_constraint(SolvedPuzzle,Puzzle,X,PreviousY,V1),
    heart_constraint(SolvedPuzzle,Puzzle,PreviousX,Y,V2),
    heart_constraint(SolvedPuzzle,Puzzle,NextX,Y,V3),
    heart_constraint(SolvedPuzzle,Puzzle,X,NextY,V4),
    V + V1 + V2 + V3 + V4 #= 10.

apply_constraint_cell(_,_,_,_,_).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%    Auxiliar Functions   %%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

check_puzzle_limits(X,Y):-
    puzzle_size(XSize,YSize),
    X #=< XSize #/\ X #> 0 #/\ Y #=< YSize #/\ Y #> 0.

%% Apply the star constraint to neighbord (X,Y)
star_constraint(SolvedPuzzle,X,Y):-
    check_puzzle_limits(X,Y),
    nth1(Y,SolvedPuzzle,Line),
    element(X,Line,V),
    V in {0,4,6,8,9}.
star_constraint(_,_,_).

%% Apply the square constraint to neighbord (X,Y)
square_constraint(SolvedPuzzle,Puzzle,X,Y,V):-
    check_puzzle_limits(X,Y),
    nth1(Y,Puzzle,L),
    nth1(X,L,Type),
    Type #\= 3,
    nth1(Y,SolvedPuzzle,Line),
    element(X,Line,V1),
    V1 #\= V.
square_constraint(_,_,_,_,_).
```

```
%% Apply the circle constraint to the cell (X,Y)
%% If a circle of the puzzle has already been applied,
%% apply the restriction of equality,
%% otherwise apply the general restriction of a circle
%% and save the position of that circle with an assert.
circle_constraint(SolvedPuzzle,X,Y):-
    circle_value(X1,Y1),
    X1 > 0,
    nth1(Y1,SolvedPuzzle,Line1),
    element(X1,Line1,V),
    nth1(Y,SolvedPuzzle,Line),
    element(X,Line,V1),
    V1 #= V.

circle_constraint(SolvedPuzzle,X,Y):-
    nth1(Y,SolvedPuzzle,Line),
    element(X,Line,V),
    V mod 3 #\= 0,
    asserta((circle_value(X,Y))).

%% Maps an identifier to the values of the
%% increment of X and Y to a knight attack position
knight_attack_range(1,1,-2).
knight_attack_range(2,2,-1).
knight_attack_range(3,2,1).
knight_attack_range(4,1,2).
knight_attack_range(5,-1,2).
knight_attack_range(6,-2,1).
knight_attack_range(7,-2,-1).
knight_attack_range(8,-1,-2).

%% Apply the knight constraint to all acttack positions
knight_constraint(_,_,_,9,0).
knight_constraint(SolvedPuzzle,X,Y,AttackId,Count):-
    knight_attack_range(AttackId,IncX,IncY),
    NewX #= X + IncX,
    NewY #= Y + IncY,
    nth1(NewY,SolvedPuzzle,Line),
    element(NewX,Line,V),
    (V mod 2 #= 0 )#<=>  C,
    NewAttackId #= AttackId + 1,
    knight_constraint(SolvedPuzzle,X,Y,NewAttackId,C1),
    Count #= C + C1.

knight_constraint(SolvedPuzzle,X,Y,AttackId,Count):-
```

```
    NewAttackId #= AttackId + 1,
    knight_constraint(SolvedPuzzle,X,Y,NewAttackId,Count)
.

%% Return the Variable in the position (X,Y) if is a
heart type,
%% otherwise return 0.
heart_constraint(SolvedPuzzle,Puzzle,X,Y,V):-
    check_puzzle_limits(X,Y),
    nth1(Y,Puzzle,TypeLine),
    element(X, TypeLine,Type),
    Type #= 7,
    nth1(Y,SolvedPuzzle,Line),
    element(X,Line,V).
heart_constraint(_,_,_,_,0).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%    Puzzle Generator   -%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

generate(XSize,YSize):-
    new_puzzle(XSize,YSize,Puzzle),
    length(LineSums,YSize),
    domain(LineSums,0,100),
    all_distinct(LineSums),
    asserta(puzzle_size(XSize,YSize)),
    new_puzzle(LineSums,SolvedPuzzle),

    asserta(circle_value(0,0)),
    apply_constraints(SolvedPuzzle,Puzzle,1,1),
    append(SolvedPuzzle,List),
    labeling([],List),
    display_game(Puzzle, SolvedPuzzle, LineSums).

new_puzzle(_,0,[]).
new_puzzle(XSize,YSize,Puzzle) :-
    new_line_types(7,XSize,Line),
    NextY is YSize - 1,
    new_puzzle(XSize,NextY,P1),
    append([Line],P1,Puzzle).

new_line_types(Limit,Length,Line):-
    length(Line,Length),
    domain(Line,0,Limit).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%    BOARD DISPLAY    %%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
display_game(Puzzle, SolvedPuzzle, Sums) :-
    nth1(1,Puzzle,Line),
    length(Line,Col),
    display_legend,
    display_puzzle(Puzzle,SolvedPuzzle,Sums,Col),nl.

display_legend :-
    nl,
    write('  S - Star\n'),
    write('  Q - Square\n'),
    write('  D - Diamond\n'),
    write('  C - Circle\n'),
    write('  T - Triangle\n'),
    write('  K - Knight\n'),
    write('  H - Heart\n\n').

/* Imprime o separador de linhas */
display_lin_separ(Col) :-
    space(1),
    write('-----'),
    N1 is Col - 1,
    display_lin_separ(N1, Col),
    write('       '),write('-----'),
    display_lin_separ(N1, Col), nl.
display_lin_separ(N, Col) :-
    N > 0, !,
    write('----'),
    N1 is N - 1,
    display_lin_separ(N1, Col).
display_lin_separ(0,_).

/* Imprime o tabuleiro */
display_puzzle([L|T],[Solved|SolvedPuzzle],[Sum|Sums] ,
Col) :-
    display_lin_separ(Col),
    write(' | '),
    display_line(puzzle,L),
    write_sum(Sum),
    write('    | '),
    display_line(solution,Solved),
    write_sum(Sum),
    new_line(1),
    display_puzzle(T, SolvedPuzzle, Sums, Col).
```

```
display_puzzle([], [], [], Col) :- dis-
play_lin_separ(Col).

/* Imprime a linha com o numero da pecas */
display_line(_,[]).
display_line(puzzle,[C|L]) :-
    write_type(C),
    write(' | '),
    display_line(puzzle,L).

display_line(solution,[C|L]) :-
    write(C),
    write(' | '),
    display_line(solution,L).

write_sum(Sum):- Sum >= 10, write(Sum).
write_sum(Sum):- Sum < 10, write(' '), write(Sum).

write_type(0):- write(' ').
write_type(1):- write('S').
write_type(2):- write('Q').
write_type(3):- write('D').
write_type(4):- write('C').
write_type(5):- write('T').
write_type(6):- write('K').
write_type(7):- write('H').
```

Utilitarios.pl

```
empty_cel(0-0).

clr :- write('\33\[2J').

wait_enter :-
    write('Press any key to continue...'),
    new_line(2),
    get_char(_).

change_player(1, 2).
change_player(2, 1).

isEmpty([]).
isEmpty([_|_]) :- !, fail.
```

```
/* Conta os elementos de uma lista */
count_lines([_|L], NumL) :-
    count_lines(L,N),
    NumL is N + 1.
count_lines([],0).

/* Calcula e retorna o tamanho do tabuleiro (Linhas e
Colunas) */
board_size([H | T], Lin, Col) :-
    count_lines(H, Col),
    count_lines(T, X),
    Lin is X + 1.

space(1) :- write(' ').
space(N) :-
    N > 1,
    write(' '),
    Next is N - 1,
    space(Next).

new_line(1) :- nl.
new_line(N) :-
    N > 1,
    nl,
    Next is N - 1,
    new_line(Next).

not(X) :- X ,! ,fail.
not(_).

/* Substitui um o elemento da posicao (Lin,Col) por Elem
*/
replace([X|L], Elem, 0, Col, [Y|L]) :-
    replace(X, Elem, Col, Y).
replace([X|L], Elem, Lin, Col, [X|NewL]) :-
    Lin > 0,
    Lin1 is Lin - 1,
    replace(L, Elem, Lin1, Col, NewL).

replace([], _, _, _).
replace([_|L], Elem, 0, [Elem|L]).
replace([X|L], Elem, Col, [X|N]) :-
    Col > 0,
    Col1 is Col - 1,
```

```
    replace(L, Elem, Col1, N).

/* Le inputs da consola e retorna se for um numero válido
*/
:- dynamic choice/1.
get_number(Choice) :-
    asserta((choice(10):-!)),
    get_code(Code1),
    between(48, 57, Code1),
    Num1 is Code1 - 48,
    asserta((choice(Num1):-!)),
    peek_code(Code2),
    between(48, 57, Code2),
    Num2 is Code2 - 48,
    Choice is Num1 * 10 + Num2,
    skip_line, !,
    retractall(choice(_)).

get_number(Choice) :-
    choice(Choice),
    Choice == 10,
    retractall(choice(_)),
    !, fail.

get_number(Choice) :-
    choice(Choice),
    skip_line,
    retractall(choice(_)).
```

puzzles.pl

```
%% PUZZLEs 7x7 with 7 types
puzzle(1, [ [2,6,0,3,2,4,0],
            [2,2,5,4,2,2,0],
            [4,0,5,3,6,2,5],
            [7,2,5,0,3,0,2],
            [7,4,3,4,1,2,2],
            [2,5,2,2,2,2,6],
            [4,5,3,0,4,2,5]], [23,20,26,20,26,19,22]).

puzzle(2, [ [4,0,3,6,6,4,7],
            [0,2,2,4,7,2,7],
            [5,2,7,3,7,2,4],
            [5,0,7,4,2,2,7],
            [5,6,1,3,2,4,7],
```

```
            [5,0,7,3,2,2,5],
            [2,2,7,3,2,4,5]], [30,43,24,31,37,27,28]).

puzzle(3, [ [1,0,4,3,2,2,2],
            [0,5,7,4,2,2,2],
            [5,5,7,3,2,4,7],
            [4,2,2,4,3,6,7],
            [7,2,3,4,7,7,2],
            [7,4,3,7,5,2,2],
            [6,0,3,7,2,2,4]], [19,29,21,18,25,39,17]).

%% PUZZLEs 4x5 and 7x7 with 5 types
puzzle(4, [ [1,4,3,0],
            [4,5,4,3],
            [0,3,5,4],
            [2,4,0,3],
            [2,2,0,3]], [15,18,23,14,14]).

puzzle(5, [ [0,3,1,4],
            [2,2,0,3],
            [2,3,5,4],
            [4,0,3,5],
            [5,3,2,1]], [28,18,21,22,12]).

puzzle(6, [ [0,4,3,2],
            [5,4,3,2],
            [5,4,2,2],
            [5,2,4,2],
            [5,3,4,2]], [18,19,10,13,3]).

puzzle(7, [ [4,4,2,3],
            [0,5,3,2],
            [1,0,4,0],
            [0,5,3,4],
            [5,4,2,3]], [18,10,26,21,18]).

puzzle(8, [ [0,3,2,2,3,4,0],
            [2,2,2,3,4,5,4],
            [0,2,3,4,1,4,2],
            [4,0,4,3,0,2,2],
            [5,3,5,3,4,0,2],
            [0,0,0,0,5,2,2],
            [5,5,5,5,4,4,4]], [19,21,28,32,11,38,40]).

puzzle(9, [ [2,2,2,3,4,2,0],
```

```
            [2,2,0,3,4,2,2],
            [4,2,2,3,4,0,2],
            [5,3,2,3,4,5,2],
            [4,2,0,2,2,2,2],
            [5,3,5,4,4,0,2],
            [2,3,4,1,0,4,2]], [20,23,24,24,18,31,24]).

%% PUZZLEs 4x4 with 4 types
puzzle(10, [[1,4,3,2],
            [0,3,2,4],
            [2,0,4,3],
            [2,4,1,0]], [19,22,14,23]).

puzzle(11, [[4,1,3,1],
            [2,4,1,2],
            [2,2,4,0],
            [2,0,1,4]], [25,6,9,10]).

puzzle(12, [[0,2,4,3],
            [4,2,2,3],
            [0,3,2,4],
            [1,0,4,4]], [18,14,20,19]).

puzzle(13, [[4,2,0,3],
            [2,2,2,2],
            [2,2,3,4],
            [0,4,4,3]], [10,10,12,18]).
```