Bosnian Road

Daniel Machado[201506365] and Sofia Alves[201504570]

[1] Faculdade de Engenharia da Universidade do Porto,
R. Dr. Roberto Frias, 4200-465 Porto
direcao.feup@fe.up.pt,
WWW home page: http://fe.up.pt
[2] Universidade do Porto,
Praça de Gomes Teixeira, 4099-002 Porto

Resumo Este trabalho surge no contexto da Unidade Curricular, Programação Lógica, tendo como objetivo utilizar a programação lógica com restrições para resolver problemas de decisão. O nosso problema tem como base o puzzle Bosnian Road, onde o objetivo é desenhar um único circuito fechado sob a condição de em volta de uma célula com o número X se encontrarem X e apenas X células pertencentes ao caminho. Utilizando os predicados fornecidos pelo *SICTus Prolog* foi possível resolver o problema de forma eficiente assim como generalizar a sua solução para casos mais gerais.

Keywords: prolog, inteligência artificial, programação em lógica

1 Introdução

Este projeto tem como objetivo principal a resolução de um problema, neste caso um puzzle, utilizando programação lógica com restrições. Para isso deve ser usada a biblioteca *'clpfd'* presente no SICStus Prolog pois fornece um conjunto de predicados essenciais neste tipo de paradigma. Após ponderar sobre os temas propostos, o nosso grupo optou pelo *puzzle* Bosnian Road. Trata-se de um problema bastante intuitivo e que se revelou fácil de perceber, o que nos permitiria ter tempo para a generalização do mesmo, conseguindo resolver *puzzles* de várias dimensões além de gerar tabuleiros aleatoriamente.

Este artigo pretende documentar a abordagem que o grupo utilizou para atingir o resultado pretendido, explicando o problema em detalhe e a sua resolução. Em primeiro lugar vamo-nos focar numa secção mais ligada à abordagem utilizada na modelação do problema, abordando: as variáveis de decisão que foram utilizadas, as restrições implementadas, a forma como cada solução é avaliada assim como a estratégia de pesquisa utilizada. De seguida vamos falar sobre a visualização da solução, nomeadamente os predicados desenvolvidos neste contexto e analisar os resultados finais assim como as conclusões que se podem retirar dos mesmos.

2 Descrição do Problema

O desafio proposto para resolução é um problema de decisão, Bosnian Road. Este *puzzle* carateriza-se por o seu tabuleiro ser quadrado, ou seja, uma matriz com o mesmo número de linhas e de colunas. Antes de ser resolvido, a maior parte das casas encontram-se vazias, porém algumas possuem um número que pode variar entre 1 e 8. Este número está diretamente ligado à resolução do puzzle. Cada casa pode ter 8 casas adjacentes (cima, baixo, direita, esquerda, cima-esquerda, cima-direita, baixo-esquerda, baixo-direita), e esse número traduz o número dessas mesmas casas adjacentes que irão ser ocupadas pelo circuito. O objetivo deste problema é desenhar esse circuito de forma a respeitar todos os dados iniciais do problema e de forma a que cada célula do circuito esteja ligada, somente e apenas, a duas outras células do mesmo. Desta forma, a solução do puzzle deve ser um único circuito fechado, uma espécie de *snake*, onde as únicas partes que se tocam é a cabeça e a cauda.

3 Abordagem

O primeiro passo na abordagem foi modelar o puzzle como um problema de restrições. O grupo empenhou-se em perceber quais as variáveis de decisão a utilizar no predicado de *labelling*, a forma mais correta de restringir essas variáveis, como expandir o problema para um caso menos específico, métodos para testar os resultados obtidos e a forma mais intuitiva de interagir com o utilizador.

3.1 Variáveis de Decisão

A solução do nosso *puzzle* passa por atribuir às restantes células do tabuleiro um estado: ora se encontram vazias, ora fazem parte do circuito. Deste modo, a variável S é a nossa única variável de decisão, caso não se trate de uma pista inicial esta variável apenas pode assumir o valor 0 ou 11, zero no caso de se encontrar vazia, 11 no caso de fazer parte o circuito, sendo este o domínio das variáveis de decisão.

3.2 Restrições

Para resolver o problema proposto desenvolveu-se um predicado chamado solve que aceita um tabuleiro inicial, o tamanho do mesmo, e retorna a solução. As restrições encontram-se todas implementadas no predicado *restrict*. Este predicado percorre todas as casas do tabuleiro alterando-as consoantes as restrições implementadas.

```
1 restrict(Board,Size,Counter):-
2   Counter =< Size*Size,
3   check_surrounded(Board,Size,Counter),
4   C is Counter+1,
```

```
5     restrict(Board,Size,C).
6
7   restrict(Board,Size,Counter):-
8     Counter =< Size*Size,
9     check_closed(Board,Size,Counter),
10    C is Counter+1,
11    restrict(Board,Size,C).
12
13  restrict(Board,Size,Counter).
```

Predicado restrict

De um modo geral são definidas duas grandes restrições: uma pista deve-se encontrar rodeada das respetivas casas do circuito; o circuito deve ser único e fechado. Estas restrições são respetivamente implementadas pelos predicados *check_ surrounded* e *check_ closed*. O predicado *check_ surrounded* analisa cada célula individualmente. Caso essa célula se trate de uma pista utiliza o predicado *get_ adjacent* para ir buscar todos os valores de todas as células que se encontram à volta da célula em questão (cima, baixo, direita, esquerda, cima-esquerda, cima-direita, baixo-esquerda, baixo-direita). Depois basta verificar se o número de células com o valor 11 é exatamente igual à pista.

```
1   check_surrounded(Board,Size,Counter):-
2     get_element(Board,Counter,E),
3     \+fd_var(E),
4     E >=2,
5     get_adjacent(Board,Counter,Size,Adjacent),
6     exactly(11,Adjacent,E).
```

Predicado *check_ surrounded*

Relativamente à restrição implementada no predicado *check_ closed* é necessário verificar que o circuito é fechado. Deste modo é verificada a valor das casas adjacentes, mas neste caso sem as diagonais, pois estas iriam levar à aceitação de circuitos inválidos. Cada casa deve, apenas e somente, possuir uma ligação a duas casas adjacentes (cima-baixo, cima-direita, cima-esquerda, etc.).

```
1   check_closed(Board,Size,Counter):-
2     get_element(Board,Counter,Element),
3     fd_var(Element),
4
5     M is Counter mod Size,
6     get_rigth(Counter,Size,R,M),
7     get_left(Counter,Size,L,M),
8     get_top(Counter,Size,T),
9     get_bot(Counter,Size,B),
10
11    get_element(Board,R,E1),
12    get_element(Board,L,E2),
13    get_element(Board,T,E3),
```

```
14    get_element(Board,B,E4),
15
16    ((Element #=11 #/\ (
17    (E1 #=11 #/\ E2#=11 #/\ E3#\=11 #/\ E4 #\=11) #\/
18    (E1 #=11 #/\ E3#=11 #/\ E2#\=11 #/\ E4 #\=11) #\/
19    (E1 #=11 #/\ E4#=11 #/\ E2#\=11 #/\ E3 #\=11) #\/
20    (E2 #=11 #/\ E3#=11 #/\ E1#\=11 #/\ E4 #\=11) #\/
21    (E2 #=11 #/\ E4#=11 #/\ E1#\=11 #/\ E3 #\=11) #\/
22    (E3 #=11 #/\ E4#=11 #/\ E1#\=11 #/\ E2 #\=11) )) #\/
23    Element #=0).
24  check_closed(Board,Size,Counter).
```

Predicado *check_ closed*

3.3 Função de Avaliação

Como o nosso *puzzle* não é um problema de otimização, não fazia sentido criar uma função de avaliação das soluções obtidas. Não sendo necessário comparar várias soluções para obter a melhor, não foi desenvolvido nada nesse sentido. Na nossa implementação a partir do momento que se cumprem todas as restrições, a solução encontrada é válida, pois já é usada alguma otimização no predicado de *labeling*. Assim sendo, a nossa forma de avaliar o desempenho do programa é através de algumas estatísticas, por exemplo o tempo decorrido, o número de *backtracks*, o número de restrições, entre outras. Estas estatísticas são mostradas ao utilizador sempre que é resolvido um *puzzle*.

```
========= Statistics =========

Time: 0.03s
Resumptions: 79856
Entailments: 78223
Prunings: 49386
Backtracks: 131
Constraints: 2833
```

Figura 1. Exemplo de estatísticas aquando a resolução de um *puzzle* 8x8

3.4 Estratégia de Pesquisa

A nossa estratégia de *labeling* - etiquetagem tenta ser o mais otimizada possível. Após a obtenção de valores que conseguem ultrapassar todas as restrições implementadas, a nossa função de etiquetagem apenas computa o melhor valor. Isto é possível através da utilização de uma combinação de opções que em conjunto resulta muito bem. Esta combinação passa for *ff*, *bissect* e *down*. A opção *ff* , ou

seja, *first fail*, faz *label* à variável mais à esquerda e com o menor domínio, para que seja possível detetar a falha o mais cedo possível. A opção *bissect* faz com que para cada variável, a escolha seja feita através do ponto médio do domínio, daí o nome bisseção. Por último, a opção *down*, tenta primeiro os elementos do domínio com o valor mais elevado.

```
solve(Board,Size,Solution):-
  initialize(Board, Solution),
  restrict(Solution,Size,1),
  labeling([ff,bisect,down], Solution).
```

Estratégia de *labeling*

4 Visualização da Solução

Para representar o tabuleiro foi utilizada uma lista inteiros. O número zero corresponde a uma casa vazia, enquanto que o número 11 corresponde a uma casa pertencente ao circuito. Além disso, para representar as pistas são utilizados os números entre 1 e 8, que representam o número de casas adjacentes á casa com a pista, que fazem parte do caminho.

```
board(3,Board,8):-
  Board = [
              0,0,0,0,0,0,0,0,
              0,7,0,4,0,3,0,0,
              0,0,0,0,0,0,0,0,
              0,4,0,0,0,0,0,0,
              0,0,0,0,0,0,0,0,
              0,0,0,0,0,0,0,0,
              0,0,0,0,0,0,4,0,
              0,0,0,0,0,0,0,0
           ].
```

Puzzle 8x8 por resolver - representação interna do tabuleiro

Na interface com o utilizador os valores das pistas permanecem os mesmos, mas os restantes são alterados para permitir uma melhor visualização da solução. O número 11 é transformado no caracter X enquanto que é zero é transformado num espaço. Esta transformação é feita através do predicado *arrange_ board*.

```
arrange_board([],[]).
arrange_board([O|Os],[N|Ns]):-
  O = 11,
  N = 'X',
  arrange_board(Os,Ns).

arrange_board([O|Os],[N|Ns]):-
  O = 0,
```

```
9     N = ' ',
10    arrange_board(Os,Ns).
11
12  arrange_board([O|Os],[N|Ns]):-
13    N = O,
14    arrange_board(Os,Ns).
```

Predicado *arrange_board*

Para imprimir o tabuleiro no ecrã é utlizado o predicado *print_board*, sendo que se recorre aos predicados *print_row_by_row*, *print_line* , *print_space_line*, *print_black_line*, para permitir um melhor alinhamento e visualização dos conteúdos do tabuleiro.

```
1   print_board(Board,Size) :-
2     Total is Size*Size,nl,
3     print_row_by_row(Board,Size,Total),
4     print_black_line(Size,0), nl.
```

Predicado *print_board*

Quando juntamos todos estes predicados com o predicado que resolve o *puzzle* em si, obtemos a seguinte solução para o *puzzle* da figura X.

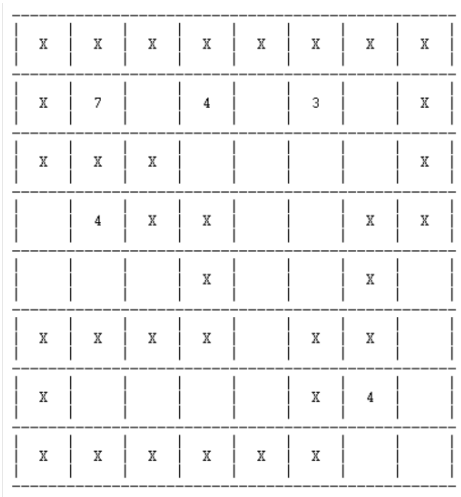

Figura 2. Solução em modo texto do *puzzle* anterior

5 Resultados

Quando um *puzzle* é resolvido, além da solução final, o nosso código também imprime no ecrã diferentes valores de estatísticas, nomeadamente: o tempo que o programa gastou a encontrar a solução, *resumptions*, *entailment*, *prunings*, *backtracks* e *constraints*.

Para recolher dados foram utilizados diferentes *puzzles*, previamente declarados no código fonte, com as dimensões 4x4, 6x6, 8x8 e 10x10. Como seria esperado, qualquer um destes valores conforme aumenta a dimensão do *puzzle*.

Dimension	Time	Resumptions	Entailment	Prunings	Backtracks	Constraints
4x4	0	885	553	736	0	513
6x6	0.01	2336	1473	1849	1	1242
8x8	0.01	79856	78223	49386	131	2833
10x10	0.03	102393	103293	89579	89	6882

6 Conclusões e Trabalho Futuro

Com a conclusão deste segundo projeto, podemos concluir que a linguagem Prolog, mais especificamente os seus módulos de resolução com restrições são bastante poderosos na resolução de uma ampla gama de questões de decisão/otimização. Consideramos que este projeto nos ajudou bastante a interiorizar o conceito de Programação Lógica com Restrições (PLR), assim como todos os conceitos que lhe estão associados: varáveis de decisão, labelling, restrições, entre outros.

Os resultados obtidos são bastante satisfatórios, facilmente se observa que com o aumento da dimensão do *puzzle* temos uma estratégia cada vez mais complexa de resolução. Estes resultados também mostram que o nosso código está bastante eficiente, uma vez que, para os exemplos apresentados não ultrapassa 1 segundo em tempo de computação (apesar de nenhum dos *puzzles* apresentar uma dimensão demasiado elevada).

Num próximo trabalho poderíamos melhorar a resolução de *puzzles* gerados aleatoriamente, uma vez que agora ainda contém algumas falhas. Mas apesar disso estamos muito confiantes no nosso projeto, pois conseguimos apresentar boas soluções para qualquer dimensão de *puzzle*.

7 Referências

- Vários autores, The Prolog Library – SICStus Prolog, https://sisctis.sics.se/sisctus/docs/4.3.0/html/sicstus/The-Prolog-Lybrary.html, documentação dos predicados disponíveis nas várias bibliotecas do SICStus Prolog
- Slides da disciplina sobre PLR, Henrique Lopes Cardoso (regente), https://moodle.up.pt/course/view.php?id=638 consultado em dezembro de 2017
- Ehud Sterling Leon, Shapiro (autor), The Art of Prolog, Second Edition: Advanced Programming Techniques (Logic Programming), 1994
- Pierre Deransart, Abdel Ed-Dbali, Laurent Cervoni (autores), Prolog: The Standard:Reference Manual, Springer, 2007
- Pagina com as Regra do Jogo, vários autores, http://logicmastersindia.com, consultado em dezembro 2017

A Código fonte

A.1 main.pl

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

:-include('utils.pl').
:-include('solve.pl').
:-include('interface.pl').
:-include('generate.pl').

start:-
  clear_screen,
  print_menu,
  read(Option),get_char(_),
  menu(Option).

menu(1):-
  clear_screen,
  print_board_menu,
  read(Option),get_char(_),
  board_menu(Option),
  read(X),get_char(_),
  start.

menu(2):-
  clear_screen,
  print_size_menu,
  read(Option),get_char(_),
  generate(Option,Board,Size),
  clear_screen,
  print_board(Board,Size),
  read(X),get_char(_),
  start.


menu(3):-
  abort.

menu(_):-
  clear_screen,
  false.

board_menu(B):-
  clear_screen,
  board(B,Board,Size),
  init_stats,
  solve(Board,Size,Solution),
  print_time,
```

```
46    print_stats,
47    arrange_board(Solution,Printable),
48    print_board(Printable,Size).
```

Main

A.2 solve.pl

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


initialize([], []).
initialize([B | Bs], [S | Ss]):-
  B =0,
  S in (0..0) \/ (11..11),
  initialize(Bs, Ss).

initialize([B | Bs], [S | Ss]):-
  S =B,
  initialize(Bs, Ss).


get_element(Board,Idx,Element):-
  nth1(Idx,Board,Element).
get_element(Board,Idx,-1).

get_element(Board, Idx, Inputs, [Element | Inputs]):-
  nth1(Idx,Board,Element).
get_element(_, _, Inputs, Inputs).

get_rigth(Counter,Size,R,0):-  R is -1.
get_rigth(Counter,Size,R,_):-  R is Counter + 1.

get_left(Counter,Size,L,1):-  L is -1.
get_left(Counter,Size,L,_):-  L is Counter - 1.

get_top(Counter,Size,T):-  T is Counter - Size.
get_bot(Counter,Size,B):-  B is Counter + Size.

get_diogonal1(Counter,Size,D,1):- D is -1.
get_diogonal1(Counter,Size,D,_):- D is Counter-Size-1.
get_diogonal2(Counter,Size,D,1):- D is -1.
get_diogonal2(Counter,Size,D,_):- D is Counter+Size-1.
get_diogonal3(Counter,Size,D,0):- D is -1.
get_diogonal3(Counter,Size,D,_):- D is Counter-Size+1.
get_diogonal4(Counter,Size,D,0):- D is -1.
get_diogonal4(Counter,Size,D,_):- D is Counter+Size+1.


get_adjacent(Board, Counter,Size, Adjacent):-
  M is Counter mod Size,
  get_rigth(Counter,Size,R,M),
  get_left(Counter,Size,L,M),
  get_top(Counter,Size,T),
  get_bot(Counter,Size,B),
  get_diogonal1(Counter,Size,D1,M),
```

```
49    get_diogonal2(Counter,Size,D2,M),
50    get_diogonal3(Counter,Size,D3,M),
51    get_diogonal4(Counter,Size,D4,M),
52
53    get_element(Board,R,[],T1),
54    get_element(Board,L,T1,T2),
55    get_element(Board,T,T2,T3),
56    get_element(Board,B,T3,T4),
57    get_element(Board,D1,T4,T5),
58    get_element(Board,D2,T5,T6),
59    get_element(Board,D3,T6,T7),
60    get_element(Board,D4,T7,Adjacent).
61
62
63  check_closed(Board,Size,Counter):-
64    get_element(Board,Counter,Element),
65    fd_var(Element),
66
67    M is Counter mod Size,
68    get_rigth(Counter,Size,R,M),
69    get_left(Counter,Size,L,M),
70    get_top(Counter,Size,T),
71    get_bot(Counter,Size,B),
72
73    get_element(Board,R,E1),
74    get_element(Board,L,E2),
75    get_element(Board,T,E3),
76    get_element(Board,B,E4),
77
78    ((Element #=11 #/\ (
79    (E1 #=11 #/\ E2#=11 #/\ E3#\=11 #/\ E4 #\=11) #\/
80    (E1 #=11 #/\ E3#=11 #/\ E2#\=11 #/\ E4 #\=11) #\/
81    (E1 #=11 #/\ E4#=11 #/\ E2#\=11 #/\ E3 #\=11) #\/
82    (E2 #=11 #/\ E3#=11 #/\ E1#\=11 #/\ E4 #\=11) #\/
83    (E2 #=11 #/\ E4#=11 #/\ E1#\=11 #/\ E3 #\=11) #\/
84    (E3 #=11 #/\ E4#=11 #/\ E1#\=11 #/\ E2 #\=11) )) #\/
85    Element #=0).
86  check_closed(Board,Size,Counter).
87
88  check_surrounded(Board,Size,Counter):-
89    get_element(Board,Counter,E),
90    \+fd_var(E),
91    E >=2,
92    get_adjacent(Board,Counter,Size,Adjacent),
93    exactly(11,Adjacent,E).
94
95
96  restrict(Board,Size,Counter):-
97    Counter =< Size*Size,
98    check_surrounded(Board,Size,Counter),
```

```
99    C is Counter+1,
100   restrict(Board,Size,C).
101
102 restrict(Board,Size,Counter):-
103   Counter =< Size*Size,
104   check_closed(Board,Size,Counter),
105   C is Counter+1,
106   restrict(Board,Size,C).
107
108 restrict(Board,Size,Counter).
109
110
111 solve(Board,Size,Solution):-
112   initialize(Board, Solution),
113   restrict(Solution,Size,1),
114   labeling([ff,bisect,down], Solution).
```

Solve

A.3 interface.pl

```
1
2  clear_screen :-
3    printLines(65).
4
5  printLines(N) :-
6      N > 0,
7      nl,
8      N1 is N - 1,
9      printLines(N1).
10 printLines(_).
11
12
13 print_menu :-
14   write('========================================\n'),
15   write('==           Bosnian Road             ==\n'),
16   write('========================================\n'),
17   write('==                                    ==\n'),
18   write('==          1 - Solve                 ==\n'),
19   write('==          2 - Generate              ==\n'),
20   write('==          3 - Exit                  ==\n'),
21   write('==                                    ==\n'),
22   write('========================================\n').
23
24
25 print_board_menu :-
26   write('========================================\n'),
27   write('==   Bosnian Road - Choose the Board  ==\n'),
28   write('========================================\n'),
29   write('==                                    ==\n'),
30   write('==          1 - Board 1 (4x4)         ==\n'),
31   write('==          2 - Board 2 (6x6)         ==\n'),
32   write('==          3 - Board 3 (8x8)         ==\n'),
33   write('==          4 - Board 4 (10x10)       ==\n'),
34   write('==                                    ==\n'),
35   write('========================================\n').
36
37 print_size_menu :-
38   write('========================================\n'),
39   write('==      Bosnian Road - Board Size     ==\n'),
40   write('========================================\n'),
41   write('==                                    ==\n'),
42   write('==             1 - 5x5                ==\n'),
43   write('==             2 - 6x6                ==\n'),
44   write('==             3 - 7x7                ==\n'),
45   write('==             4 - 8x8                ==\n'),
46   write('==                                    ==\n'),
47   write('========================================\n').
48
```

```
49  print_black_line(Size,Size):-
50    write('-'),nl.
51
52  print_black_line(Size,N):-
53    write('------'),
54    N1 is N+1,
55    print_black_line(Size,N1).
56
57  print_space_line(Size,Size):-
58    write('|'),nl.
59
60  print_space_line(Size,N):-
61    write('|     '),
62    N1 is N+1,
63    print_space_line(Size,N1).
64
65  print_board(Board,Size) :-
66    Total is Size*Size,nl,
67    print_row_by_row(Board,Size,Total),
68    print_black_line(Size,0), nl.
69
70  print_row_by_row(_,_,0).
71  print_row_by_row(Board,Size,Total):-
72    print_black_line(Size,0),
73    print_space_line(Size,0),
74    write('|  '),
75    print_line(Board,Size,Rest),nl,
76    print_space_line(Size,0),
77    T is Total-Size,
78    print_row_by_row(Rest,Size,T).
79
80  print_line(T,0,T).
81  print_line([H|T],Size,Rest):-
82    write(H), write('  |  '),
83    S is Size-1,
84    print_line(T,S,Rest).
85
86  arrange_board([],[]).
87  arrange_board([O|Os],[N|Ns]):-
88    O = 11,
89    N = 'X',
90    arrange_board(Os,Ns).
91
92  arrange_board([O|Os],[N|Ns]):-
93    O = 0,
94    N = ' ',
95    arrange_board(Os,Ns).
96
97  arrange_board([O|Os],[N|Ns]):-
98    N = 0,
```

```
99    arrange_board(Os,Ns).
100
101 print_time:-
102   statistics(runtime,[_,T]),
103     TS is ((T//10)*10)/1000,
104   nl, nl,
105   write('========= Statistics ========='), nl,
106     nl, write('Time: '), write(TS), write('s'),nl.
107
108
109 print_stats:-
110   fd_statistics(resumptions, Resumptions),
111   fd_statistics(entailments, Entailments),
112   fd_statistics(prunings, Prunings),
113   fd_statistics(backtracks, Backtracks),
114   fd_statistics(constraints, Constraints),
115   write('Resumptions: '), write(Resumptions), nl,
116   write('Entailments: '), write(Entailments), nl,
117   write('Prunings: '), write(Prunings), nl,
118   write('Backtracks: '), write(Backtracks), nl,
119   write('Constraints: '), write(Constraints), nl.
```

Interface

A.4 generate.pl

```
:-use_module(library(clpfd)).
:-use_module(library(random)).

random_numbers([],0).
random_numbers([H|T],N):-
  random(1,8,H),
  N1 is N-1,
  random_numbers(T,N1).



generate(Option,Board,Size):-
  Option >0,
  Option <5,
  Size is Option+3,
  Total is Size*Size,
  N is floor(Size*0.7),
  Z is Total-N,
  length(Zeros,Z),
  domain(Zeros,0,0),
  labeling([],Zeros),
  random_numbers(Numbers,N),
  append(Zeros,Numbers,Tmp),
  random_permutation(Tmp,Board).

generate(_,_,_):-
  clear_screen,
  false.
```

Generate

A.5 utils.pl

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


init_stats :-
 statistics(runtime,_).

add_list([H|O],O,H).

board(1,Board,4):-
    Board = [
        0,0,0,0,
        0,0,7,0,
        0,0,0,0,
        2,0,0,0
         ].


board(2,Board,6):-
  Board = [
      0,0,0,0,0,0,
      0,5,0,0,5,0,
      0,0,3,0,0,0,
      0,0,0,0,0,0,
      0,3,0,0,0,0,
      0,0,0,0,0,2
      ].

board(3,Board,8):-
  Board = [
             0,0,0,0,0,0,0,0,
             0,7,0,4,0,3,0,0,
             0,0,0,0,0,0,0,0,
             0,4,0,0,0,0,0,0,
             0,0,0,0,0,0,0,0,
             0,0,0,0,0,0,0,0,
             0,0,0,0,0,0,4,0,
             0,0,0,0,0,0,0,0
          ].

board(4,Board,10):-
  Board = [
             0,0,0,0,0,0,0,0,0,2,
             0,6,0,0,0,0,0,0,0,0,
             0,0,0,0,0,0,5,0,0,0,
             0,0,0,0,0,0,0,0,0,0,
             0,0,0,0,0,0,0,0,6,0,
             0,6,0,0,0,0,0,0,0,0,
             0,0,0,0,0,0,4,0,0,0,
```

```
49                0,0,0,4,0,0,0,0,0,0,
50                0,0,0,0,0,0,0,0,7,0,
51                0,0,0,0,0,5,0,0,0,0
52                          ].
53
54
55  board(_,_,_):-false.
56
57
58  exactly(_,[],0).
59  exactly(X,[Y|L],N) :-
60  X #= Y #<=> B,
61  N #= M+B,
62  exactly(X,L,M).
```