Programação em Lógica com Restrições

Resolução de um problema “Doppelblock” em Prolog

FEUP-PLOG, Turma 3MIEIC01, Grupo Doppelblock\_4

Bruno Piedade - up201505668

Danny Soares - up201505509

Universidade do Porto, 2017/2018

**Abstract.** O objetivo deste trabalho era resolver um problema de decisão relacionado com a geração e solução do problema “Doppelblock”, utilizando programação com restrições em Prolog, da forma mais eficiente possível, evitando backtracking. Para isso desenvolvemos uma aplicação que permite resolver o problema, visualizar a resolução e verificar a sua complexidade temporal.

1. Introdução

Este trabalho foi proposto na unidade curricular “Programação em Lógica” do 3º ano do Mestrado Integrado em Engenharia Informática e Computação da Faculdade de Engenharia da Universidade do Porto, com o objetivo de desenvolver um programa em Prolog, com restrições, para a resolução de problemas de decisão ou de otimização sugeridos.

Neste caso, tratámos o problema “Doppelblock”, que é um puzzle numa grelha quadrada. Como tal, o nosso programa consiste num conjunto de predicados que permite gerar e resolver puzzles “Doppelblock” com tamanho variável, bem como a visualização da complexidade temporal da execução em função das dimensões da grelha.

Neste relatório, será feita uma descrição do problema e da nossa abordagem para o resolver. Além disso, será descrita a forma de visualização da solução e serão mostrados os resultados obtidos. No final, serão apresentadas as nossas conclusões em relação ao projeto desenvolvido.

1. Descrição do Problema

O problema *“*Doppelblock” consiste num puzzle resolvido numa grelha quadrada, com N linhas e colunas. A grelha começa completamente vazia, com um número correspondente a cada linha e a cada coluna, no exterior da grelha. Para resolver o problema, colocam-se os números de 1 até N-2 e 2 quadrados pretos em cada linha e em cada coluna, por forma a que a soma dos números entre os 2 quadrados pretos seja igual ao número que se encontra no exterior da grelha. No final a grelha fica completamente preenchida de acordo com as seguintes regras:

1. Numa mesma linha ou coluna não devem haver números repetidos.
2. Em cada linha e em cada coluna devem haver dois quadrados pretos.
3. A soma dos números entre 2 quadrados pretos da mesma linha ou coluna deve ser igual ao número no exterior da grelha correspondente a essa linha ou coluna.
4. Abordagem para solução do puzzle

A abordagem para resolver o problema, de uma forma eficiente, consistiu na determinação das variáveis de decisão, restrições e estratégia de pesquisa mais adequadas.

* 1. Variáveis de Decisão

As variáveis de decisão, neste caso, são as células da grelha quadrada. Portanto, para uma grelha de N linhas e colunas, a lista das variáveis de decisão corresponde às posições das células na grelha. Para a representação em Prolog da grelha utiliza-se uma lista de listas, em que cada uma das listas representa uma linha do tabuleiro e cada elemento dessas listas representa uma célula da grelha. Assim, a grelha é representada por uma lista com *N* listas de comprimento *N*. No estado inicial, a grelha encontra-se vazia, portanto as células da grelha aparecem “vazias”, não tendo nenhum símbolo. No estado final, as células são ocupadas por “0”, que representam os quadrados pretos, ou números de 1 até *N*-2, que representam os números que estão nas células. Como tal, o domínio das variáveis de decisão vai ser [0, *N*-2].

* 1. Restrições

As restrições descritas são aplicadas a cada linha e coluna da matriz através dos predicados *restrictRows(+Matrix, +Rows, +DiffValues, +DomainMax, +Cardinality)* e *restrictColumns(+Matrix,+Columns,+DiffValues,+DomainMax,+Cardinality)* respetivamente em que *Matrix* corresponde à matriz gerada, *Rows*/*Columns* aos índices que definem as somas, *DiffValues* ao número de valores distintos (*N*-1), *DomainMax*. ao valor máximo do domínio (*N*-2) e *Cardinality* à lista com a cardinalidade dos elementos.

### Restrição 1 – restringir a cardinalidade

Para garantir que por cada linha existiam dois 0’s e que os restantes elementos eram distintos e estavam entre a gama 1 a *N*-2 foi utilizado o predicado *global\_cardinality/2* cujos argumentos são gerados de acordo com o tamanho da matriz a partir do predicado *createCardinalityRestraints(+DomainMax, -Cardinality)* em que o *DomainMax* corresponde ao valor máximo do domínio (*N*-2) e a *Cardinality* é a lista de retorno que contém a cardinalidade de cada valor.

### Restrição 2 – soma entre blocos é igual a índice

Para garantir que a soma entre os “blocos” é igual ao índice indicado foi utilizado um autómato com recurso a um contador aplicando o predicado *automaton/8*. O autómato é definido por 3 estados - q0, q1, q2 - em que q0 corresponde ao estado inicial e q2 ao estado de aceitação. Para o estado q0 as transições possíveis são δ(q0,0,q1) para indicar que um bloco foi encontrado e δ(q0,t,q0), t ∈ [1,N-2] que corresponde a aceitar todas as transições que não sejam blocos. Para o estado q1 as transições são semelhantes exceto que em cada é incrementado ao contador (com valor inicial 0) o valor da transição contabilizando assim a soma entre os dois “blocos”. Desta forma, as transições são δ(q1,0,q2,C) e δ(q1,t,q1,C+t), t ∈ [1,N-2] em que C corresponde ao contador. Por fim, para o estado q2 as transições são δ(q2,t,q2), t ∈ [1,N-2] que corresponde a aceitar todos os valores diferentes de 0.

Todos os arcos (transições) são geradas de acordo com o tamanho da matriz através do predicado createSolveArcs(+DomainMax, +C, -Arcs) em que o DomainMax corresponde ao valor máximo do domínio (N-2), C ao contador e Arcs à lista de retorno que contém todos os arcos gerados.

* 1. Estratégia de Pesquisa

Para resolver os puzzles, criámos o predicado *doppelblock(+N,+Rows,+Columns +Generate, -Res)*, que recebe o tamanho da grelha, uma lista com as somas das linhas, uma lista com as somas das colunas e retorna o puzzle resolvido.

O *labeling* da grelha é feito linha a linha, utilizando as opções *bisect* e *down* do *labeling*. A combinação destas duas opções foi a que apresentou melhores resultados em termos de tempo de execução.

1. Abordagem para geração do puzzle

A geração eficiente de puzzles é um problema diferente da resolução dos puzzles.

Para gerar puzzles válidos, a nossa abordagem foi escolher 1 aleatório de uma lista com puzzles válidos. Para criar a lista com puzzles válidos, o predicado *getRandomDoppel/2* recorre ao predicado *find\_n/5*, que vai criar a lista e chamar o nosso predicado de resolução do puzzle, *doppelblock/5*, sem as linhas e colunas instanciadas e com a flag *Generate* ativa, para dar puzzles com solução. A ativação da flag implica a utilização de um autómato semelhante ao descrito anteriormente, ao qual foi adicionado um estado que obriga a existência de pelo menos um valor entre os “blocos” cujos arcos são construídos a partir do predicado createGenerateArcs/3. Depois, da lista são recolhidos 10 dos primeiros 10^(*N*-3) resultados distribuídos uniformemente e desses é escolhido 1 aleatoriamente para ser apresentado ao utilizador.

1. Visualização da Solução

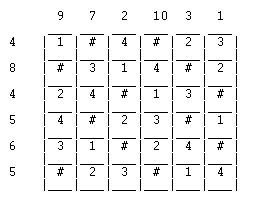
A visualização da grelha em modo de texto é feita com recurso ao predicado *printBoard/2*, que utiliza predicados auxiliares para exibir a grelha.

O predicado *printBoard/2* imprime a grelha com o aspeto de uma grelha real, com as células delimitadas lateralmente por ‘|’ e verticalmente por ‘\_’. Como o tamanho da grelha não é fixo, temos um predicado *printBorder(+Init,+Separator,+Times)* que imprime a string “Init” uma vez e depois imprime a string “Separator” “Times” vezes, o que permite desenhar os limites da grelha para qualquer tamanho.

As linhas da grelha são impressas com o predicado *p\_m(+Matriz,+Counter)* que usa o predicado *p\_l(+Linha,+Length)* para imprimir todas as “Linhas” da “Matriz” que representa a grelha.

Os números nas células representam o próprio número, enquanto “#” representa o quadrado preto.

Um possível estado da grelha será então:

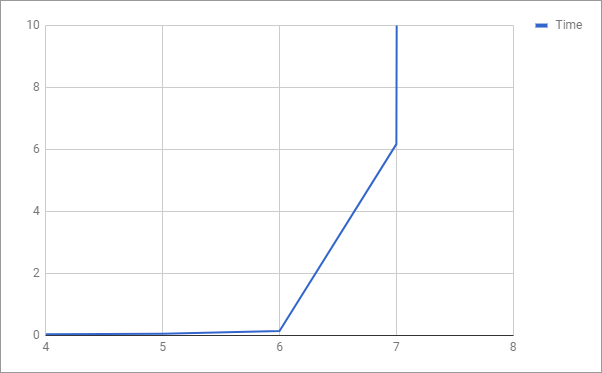


1. Resultados

A aplicação desenvolvida permite gerar grelhas aleatórias e válidas e permite solucionar estas mesmas, ou outras sugeridas pelo utilizador, que sejam válidas.

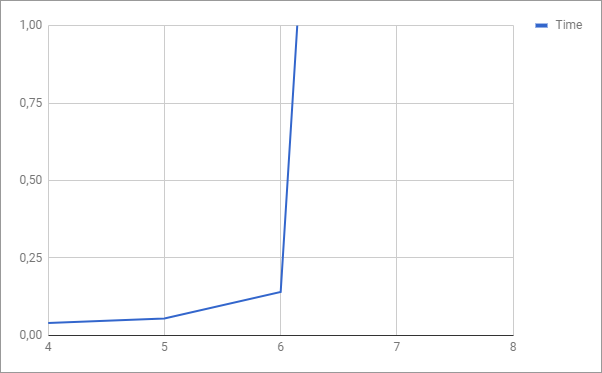
Para melhor análise dos resultados, fizemos alguns gráficos com o tempo de resolução de puzzles de tamanhos entre 4x4 e 8x8.

Na figura 1 apresentamos os valores médios da resolução de puzzles usando as opções de *labeling* *bisect* e *down*.

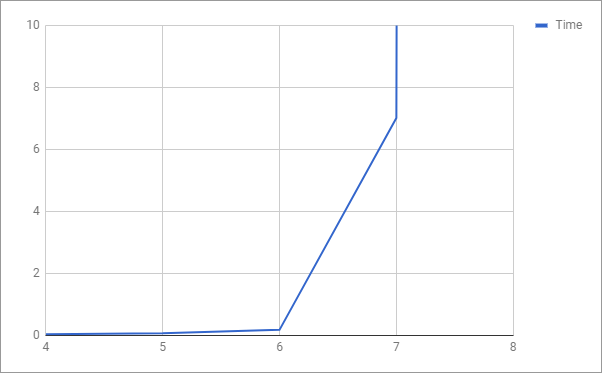
****

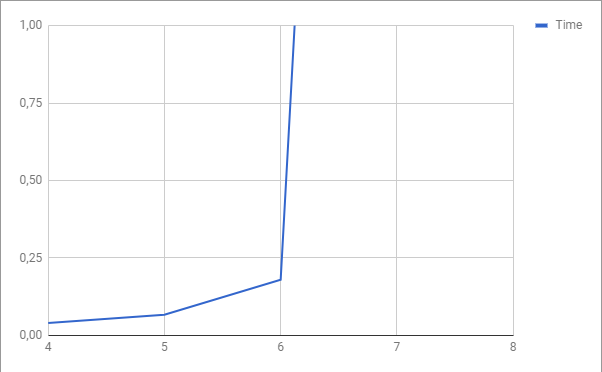
**Fig. 1.** Valores médios da resolução de puzzles de tamanhos entre 4x4 e 8x8, com *bisect* e *down* (tempo[0,10]).

Na figura 2 apresentamos o mesmo gráfico da figura 1, focando o tempo entre 0 e 1 segundos, para observar melhor a variação para grelhas de tamanhos 4 e 5, onde é possível ver que para grelhas pequenas (4x4 até 6x6) a resolução é praticamente instantânea, demorando apenas centésimos de segundo.

**Fig. 2.** Valores médios da resolução de puzzles de tamanhos entre 4x4 e 8x8, com *bisect* e *down* (tempo[0,1]).

Para comparar o tempo de execução usando outras opções de *labeling* fizemos dois gráficos semelhantes aos anteriores, utilizando outras combinações de opções. Os resultados para grelhas pequenas (4x4 a 6x6) foram bastante semelhantes, sendo a diferença de apenas uns centésimos de segundo. Para grelhas de tamanho 7x7 a diferença alcança os décimos de segundo e para grelhas de tamanho 8x8 a diferença alcança os segundos.

**Fig. 3.** Valores médios da resolução de puzzles de tamanhos entre 4x4 e 8x8, sem *bisect* e *down* (tempo[0,10]).



**Fig. 4.** Valores médios da resolução de puzzles de tamanhos entre 4x4 e 8x8, sem *bisect* e *down* (tempo[0,1]).

1. Conclusões e Trabalho Futuro

Após concluir o trabalho, achamos que o algoritmo de geração de puzzles pode ser melhorado, de forma a apresentar puzzles mais complexos.

Em relação à solução de puzzles, o algoritmo implementado mostrou-se especialmente eficiente para puzzles de tamanho inferior ou igual a “6x6”. Para grelhas maiores o algoritmo é mais lento, mas dentro de valores de tempo esperados.

Bibliografia

1. http://logicmastersindia.com/lmitests/dl.asp?attachmentid=659&view=1

Anexos

**- Ficheiro** *doppelblock.pl*

*--------------------------------------------------------------------------*

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

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

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

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

:-include('menu.pl').

:-include('utils.pl').

:-include('doppel.pl').

%GENERATE

generateDoppel(4,Res):-

length(Rows,4),

length(Columns,5),

doppelblock(4,Rows,Columns,false,Sol),

last(CleanCols,\_,Columns),

Res = [4,Rows,CleanCols,Sol].

generateDoppel(N,Res):-

ColsLength is N+1,

length(Rows,N),

length(Columns,ColsLength),

doppelblock(N,Rows,Columns,true,Sol),

last(CleanCols,\_,Columns),

Res = [N,Rows,CleanCols,Sol].

getRandomDoppel(N,Doppel):-

ResultsScale is N-3,

FilterScale is N-4,

Results is round(exp(10,ResultsScale)),

Filter is round(exp(10,FilterScale)),

find\_n(Results,Filter,Res,generateDoppel(N,Res),Doppels),

random\_member(Doppel,Doppels).

%SOLVE

createMatrix(N, N, []).

createMatrix(N, RowIdx, [Row|OtherRows]):-

length(Row,N),

NextRowIdx #= RowIdx + 1,

createMatrix(N, NextRowIdx, OtherRows).

defNDomain(N, Matrix):-

Max #= N-2,

defDomain(Max, Matrix).

defDomain(\_, []).

defDomain(Max, [H|T]):-

domain(H,0,Max),

defDomain(Max, T).

labelMatrix([],[]).

labelMatrix([H|T],[H|T2]):-

labeling([bisect, down],H),

labelMatrix(T,T2).

%AUTOMATON

createArc(Src, Dest, Val, true, Counter, Arc):-

Arc = arc(Src,Val,Dest,[Counter+Val]).

createArc(Src, Dest, Val, false,\_,Arc):-

Arc = arc(Src,Val,Dest).

createNArcs(Src, Dest, Max, Max, WithCounter, Counter, [Last]):-

createArc(Src,Dest,Max,WithCounter,Counter,Last).

createNArcs(Src, Dest, Max, Curr, WithCounter,Counter,[Arc|Others]):-

createArc(Src,Dest,Curr,WithCounter,Counter,Arc),

Next #= Curr + 1,

createNArcs(Src,Dest,Max,Next,WithCounter,Counter,Others).

createNArcs(Src, Dest, Max, Curr, WithCounter,Counter,[Arc|Others]):-

createArc(Src,Dest,Curr,WithCounter,Counter,Arc),

Next #= Curr + 1,

createNArcs(Src,Dest,Max,Next,WithCounter,Counter,Others).

createArcs(Src, Dest, Max, WithCounter, Counter, Arcs):-

createArc(Src,Dest,0,false,\_,ToDestArc),

ArcToDest = [ToDestArc],

createNArcs(Src, Src, Max, 1, WithCounter,Counter,SelfArcs),

append(ArcToDest, SelfArcs, Arcs).

createSolveArcs(Max,Counter,Arcs):-

createArcs(q0,q1,Max,false,\_,Q0Arcs),

createArcs(q1,q2,Max,true,Counter,Q1Arcs),

createNArcs(q2,q2,Max,1,false,\_,Q2Arcs),

append(Q0Arcs,Q1Arcs,TmpArcs),

append(TmpArcs,Q2Arcs,Arcs), !.

createGenerateArcs(Max,Counter,Arcs):-

createArcs(q0,q1,Max,false,\_,Q0Arcs),

createNArcs(q1,q2,Max,1,true,Counter,Q1Arcs),

createArcs(q2,q3,Max,true,Counter,Q2Arcs),

createNArcs(q3,q3,Max,1,false,\_,Q3Arcs),

append(Q0Arcs,Q1Arcs,TmpArcs),

append(TmpArcs,Q2Arcs,TmpArcs2),

append(TmpArcs2,Q3Arcs,Arcs), !.

%CARDINALITY

createCardinalityRestraints(MaxDomain,MaxDomain,[Card]):-

Card = MaxDomain-1.

createCardinalityRestraints(0,MaxDomain,[Card|Others]):-

Card = 0-2,

createCardinalityRestraints(1,MaxDomain,Others).

createCardinalityRestraints(Val,MaxDomain,[Card|Others]):-

Card = Val-1,

NextVal #= Val+1,

createCardinalityRestraints(NextVal,MaxDomain,Others).

createCardinalityRestraints(DomainMax, Cardinality):-

createCardinalityRestraints(0,DomainMax,Cardinality).

%RESTRICTIONS

restrictLine(Vars, Max, Sum, false):-

createSolveArcs(Max,C,Arcs),

automaton(Vars, \_, Vars, [source(q0), sink(q2)], Arcs, [C], [0], [Sum]).

restrictLine(Vars, Max, Sum, true):-

createGenerateArcs(Max,C,Arcs),

automaton(Vars, \_, Vars, [source(q0), sink(q3)], Arcs, [C], [0], [Sum]).

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

restrictRows([Row|OtherRows], [Value|OtherValues], DiffValues, DomainMax, Cardinality, Generate):-

global\_cardinality(Row, Cardinality),

restrictLine(Row,DomainMax,Value,Generate),

restrictRows(OtherRows,OtherValues,DiffValues,DomainMax,Cardinality,Generate).

restrictColumns(Matrix, Values, DiffValues, DomainMax,Cardinality,Generate):-

restrictColumnsIdx(Matrix,1,Values,DiffValues,DomainMax,Cardinality,Generate).

restrictColumnsIdx(\_,\_,[\_|[]],\_,\_,\_,\_).

restrictColumnsIdx(Matrix, ColIndex, [Value|OtherValues], DiffValues, DomainMax, Cardinality, Generate):-

maplistelem(ColIndex,Matrix,Col),

global\_cardinality(Col,Cardinality),

restrictLine(Col,DomainMax,Value,Generate),

NextIdx #= ColIndex + 1,

restrictColumnsIdx(Matrix, NextIdx,OtherValues,DiffValues,DomainMax,Cardinality,Generate).

doppelblock(N,Rows,Columns,Generate,Res):-

createMatrix(N, 0, Matrix),

defNDomain(N,Matrix),

DiffValues #= N-1,

DomainMax #= N-2,

createCardinalityRestraints(DomainMax,Cardinality),

restrictRows(Matrix,Rows,DiffValues,DomainMax,Cardinality,Generate),

restrictColumns(Matrix,Columns,DiffValues,DomainMax,Cardinality,Generate),

reset\_timer,

labelMatrix(Matrix,Res).

**- Ficheiro** *doppel.pl*

*--------------------------------------------------------------------------*

createDoppel(Size,Rows,Columns,Doppel):-

Doppel = [Size,Rows,Columns,\_].

getDoppelMatrix(Doppel,Matrix):-

selectAtIndex(Doppel,4,Matrix).

getDoppelColumns(Doppel,Columns):-

selectAtIndex(Doppel,3,Columns).

getDoppelRows(Doppel,Rows):-

selectAtIndex(Doppel,2,Rows).

getDoppelSize(Doppel,Size):-

selectAtIndex(Doppel,1,Size).

**- Ficheiro** *menu.pl*

*--------------------------------------------------------------------------*

doppelblock:-

clearScreen,

mainMenu.

/\*

\* MAIN MENU

\*/

mainMenu:-

write('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'),nl,

write('\*\*\*\*\*Doppelblock\*\*\*\*\*'),nl,

write('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'),nl,

write('\* \*'),nl,

write('\* Main Menu \*'),nl,

write('\* \*'),nl,

write('\* 1 - Play \*'),nl,

write('\* \*'),nl,

write('\* 0 - Exit \*'),nl,

write('\* \*'),nl,

write('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'),nl,

write('\*Option: \*'),nl,

readOption(Option),

write(Option),nl,

integer(Option), Option >= 0, Option < 2, !,

mainMenuOption(Option).

mainMenu:-

clearScreen,

write('Error: invalid input.'), nl,

mainMenu.

mainMenuOption(0):- !.

mainMenuOption(1):-

clearScreen,

playMenu.

/\*

\* PLAY MENU

\*/

playMenu:-

write('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'),nl,

write('\*\*\*\*\*\*\*\*Doppelblock\*\*\*\*\*\*\*'),nl,

write('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'),nl,

write('\* \*'),nl,

write('\* Play Menu \*'),nl,

write('\* \*'),nl,

write('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'),nl,

write('\* \*'),nl,

write('\*Choose board size (4-8)\*'),nl,

write('\*Option: \*'),nl,

readOption(Size),

integer(Size), Size >= 4, Size < 9, !,

clearScreen,

rcMenu(Size).

playMenu:-

clearScreen,

write('Error: invalid input.'), nl,

playMenu.

/\*

\* RC MENU

\*/

rcMenu(Size):-

write('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'),nl,

write('\*\*\*\*\*\*\*\*Doppelblock\*\*\*\*\*\*\*'),nl,

write('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'),nl,

write('\* \*'),nl,

write('\* Play Menu \*'),nl,

write('\* \*'),nl,

write('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'),nl,

write('\* \*'),nl,

write('\*Do you want to choose \*'),nl,

write('\*rows and columns sums? \*'),nl,

write('\*Option (1-yes 0-no): \*'),nl,

readOption(Option),

integer(Option), Option >= 0, Option < 2, !,

rcMenuOption(Option,Size).

rcMenu(Size):-

clearScreen,

write('Error: invalid input.'), nl,

rcMenu(Size).

rcMenuOption(0,Size):-

nl,nl,write('Generating...'),

getRandomDoppel(Size,Doppel),

clearScreen,

nl,write('Randomly generated puzzle: '),nl,nl,

solveMenu(Doppel).

rcMenuOption(1,Size):-

getRows(Rows,Size),

verifyInts(Rows),

verifySums(Rows,Size),

getCols(Cols,Size),

verifyInts(Cols),

verifySums(Cols,Size),!,

clearScreen,

nl, write('Selected puzzle: '), nl,nl,

createDoppel(Size,Rows,Cols,Doppel),

createClearMatrix(Size,Matrix),

printMatrix(Rows,Cols,Matrix),nl,

nl, write('Press Enter to solve'), nl,

waitForEnter,

waitForEnter,

solveMenuOption(1,Doppel).

rcMenuOption(1,Size):-

write('Error: array must have only integers less than the sum of all integers from 1 to Size-2.'),nl,

rcMenuOption(1,Size).

solveMenu(Doppel):-

getDoppelSize(Doppel,Size),

getDoppelRows(Doppel,Rows),

getDoppelColumns(Doppel,Columns),

createClearMatrix(Size,Matrix),

printMatrix(Rows,Columns,Matrix),nl,

nl,nl,

write('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'),nl,

write('\* \*'),nl,

write('\* 1 - Solve \*'),nl,

write('\* 2 - Show solution \*'),nl,

write('\* \*'),nl,

write('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'),nl,

readOption(Option),

integer(Option), Option > 0, Option < 3, !,

solveMenuOption(Option,Doppel).

solveMenu(Doppel):-

clearScreen,

write('Error: invalid input.'), nl,

solveMenu(Doppel).

solveMenuOption(1,Doppel):-

getDoppelSize(Doppel,Size),

getDoppelRows(Doppel,Rows),

getDoppelColumns(Doppel,Columns),

nl, write('Solving...'),

doppelblock(Size,Rows,Columns,false,Matrix),

clearScreen,

printSolutionText,

printMatrixWithStats(Rows,Columns,Matrix),

nl, write('Press Enter to continue...'),

waitForEnter,

clearScreen,

mainMenu.

solveMenuOption(2,Doppel):-

getDoppelRows(Doppel,Rows),

getDoppelColumns(Doppel,Columns),

getDoppelMatrix(Doppel,Matrix),

clearScreen,

printSolutionText,

printMatrix(Rows,Columns,Matrix),

nl, write('Press Enter to continue...'),

waitForEnter,

clearScreen,

mainMenu.

printSolutionText:-

nl,nl,

write(' \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'),nl,

write(' \* SOLUTION \*'),nl,

write(' \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'),nl,

nl,nl,nl.

**- Ficheiro** *utils.pl*

*--------------------------------------------------------------------------*

:- dynamic(find\_n\_solution/1).

:- dynamic(find\_n\_counter/1).

%CONVERTS the list symbol to the board symbol

convert(0,'#').

convert(X,X).

%IF\_THEN\_ELSE

ite(If,Then,\_):- If, !, Then.

ite(\_,\_,Else):- Else.

%SELECT\_AT\_INDEX

selectAtIndex(List, Index, Elem):-

nth1(Index, List, Elem).

%SELECT\_POS

selectPos(State,X,Y,Elem):-

nth1(Y,State,Row),

nth1(X,Row,Elem).

%CLEAR\_SCREEN

clearScreen:-

write('\33\[2J').

%USER\_I/O

%CONVERT\_ASCII\_CODE\_TO\_NUMBER

codeToNumber(Code,Value):-

Value is Code-48 .

%READ\_STRING

readString([Char|OtherChars]):-

get\_code(Char),

ite(Char = 10, (OtherChars = [],true), readString(OtherChars)).

%READ\_MENU\_OPTION

readOption(Option):-

readString(String),

selectAtIndex(String,1,OptionCode),

codeToNumber(OptionCode,Option).

readArray(Array):-

read(Array).

getRows(Rows,Size):-

write('Rows sums ([R1,R2,R3,...]) '),

write('Size = '),write(Size),write(':'),nl,

readArray(Rows),

length(Rows, Size),!.

getRows(Rows,Size):-

write('Error: wrong array size.'), nl,

getRows(Rows,Size).

getCols(Cols,Size):-

write('Columns sums ([C1,C2,C3,...]) '),

write('Size = '),write(Size),write(':'),nl,

readArray(Cols),

length(Cols, Size),!.

getCols(Cols,Size):-

write('Error: wrong array size.'), nl,

getCols(Cols,Size).

verifyInts([]).

verifyInts([H|T]):-

integer(H),

verifyInts(T).

verifySums([],\_).

verifySums([H|T],Size):-

Sum is ((Size-2)\*(Size-1)/2),

H =< Sum,

verifySums(T,Size).

%WAIT\_FOR\_ENTER

waitForEnter:-

readString(\_).

%PRINT\_MATRIX

printVal(X):-

X < 10,

write(X),write(' ').

printVal(X):-

write(X),write(' ').

printColumns(Columns):-

write(' '),

maplist(printVal,Columns), nl.

printBorderTimes(\_, Times, Times).

printBorderTimes(Separator, Curr, Times):-

write(Separator),

Next is Curr+1,

printBorderTimes(Separator, Next, Times).

printBorder(Init, Separator, Times):-

write(Init),

printBorderTimes(Separator, 0, Times),

nl.

p\_m([],\_).

p\_m([L|T],[Row|Rows]):-

printVal(Row),

proper\_length(L,Length),

p\_l(L,Length),

p\_m(T,Rows).

p\_l([C|[]],Length):- convert(C,S),write('| '), write(S), write(' |'), nl, printBorder(' |','\_\_\_|',Length).

p\_l([C|T],Length):- convert(C,S),write('| '), write(S), write(' '), p\_l(T,Length).

printMatrix(Rows,Columns,Matrix):-

proper\_length(Matrix,Length),

printColumns(Columns),

printBorder(' ', '\_\_\_ ',Length),

p\_m(Matrix,Rows),!.

printMatrixWithStats(Rows,Columns,Matrix):-

printMatrix(Rows,Columns,Matrix),

print\_time,

fd\_statistics.

%STATISTICS

reset\_timer :- statistics(walltime,\_).

print\_time :-

statistics(walltime,[\_,T]),

TS is ((T//10)\*10)/1000,

nl, write('Time: '), write(TS), write('s'), nl, nl.

%CREATE\_CLEAR\_MATRIX

clearVal(X):-

X = ' '.

clearLine(N,Line):-

length(Line, N),

maplist(clearVal,Line).

createClearMatrix(N,Matrix):-

length(Matrix,N),

maplist(clearLine(N), Matrix).

%MAP\_LIST\_ELEM

maplistelem(Pos, Xs, Ys) :-

( foreach(X,Xs),

foreach(Y,Ys),

param(element)

do call(element, Pos, X, Y)

).

%FIND\_N\_SOLUTIONS

find\_n(N, Filter, Term, Goal, Solutions) :-

( set\_find\_n\_counter(N),

retractall(find\_n\_solution(\_)),

once((

call(Goal),

dec\_find\_n\_counter(M),

/\*write(M),nl,\*/

Sol is mod(M,Filter),

ite(Sol == 0,

assertz(find\_n\_solution(Term)),

true

),

M =:= 0

)),

fail

; findall(Solution, retract(find\_n\_solution(Solution)), Solutions)

).

set\_find\_n\_counter(N) :-

retractall(find\_n\_counter(\_)),

assertz(find\_n\_counter(N)).

dec\_find\_n\_counter(M) :-

retract(find\_n\_counter(N)),

M is N - 1,

assertz(find\_n\_counter(M)).