Magic Snail
Resolução de um Problema de Decisão usando Programação em Lógica com Restrições

Turma 3MIEIC1
Grupo Magic Snail_4
Leonardo Manuel Gomes Teixeira – up201502848
Maria Eduarda Santos Cunha – up201506524

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

Índice

1 Resumo

Este projeto surgiu da necessidade de complementar e consolidar os conceitos lecionados nas aulas práticas e teóricas da cadeira de Programação em Lógica, mais especificamente os problemas de decisão ou otimização com restrições.

Optamos pela implementação de um programa em Prolog que resolvesse o jogo Magic Snail. Este jogo tem por base um tabuleiro quadrado, em que a ordem das células respeita a forma de um caracol e o objetivo é preenchê-las com letras de forma a que a sua ordem, quando lidas, corresponda à chave de jogo, sem que haja repetição de nenhum elemento da chave ao longo de uma coluna ou linha.

2 Introdução

Face os temas propostos na cadeira de Programação Lógica para o projeto final, que se podiam incluir na categoria de problema de decisão ou otimização, decidimos trabalhar o jogo Magic Snail. Este jogo insere-se nos problemas de decisão e o objetivo principal era a aplicação de conceitos de programação lógica com restrições.

O Magic Snail é um jogo de tabuleiro quadrado em que as células são lidas como se este se tratasse da carapaça de um caracol. O objetivo é que os jogadores preencham as células de forma a que, quando lidas, se obtenha sempre a chave de jogo. Ainda, nenhum elemento da chave se deve repetir na mesma linha ou coluna.

A nossa abordagem passou por estabelecer como queríamos definir o tabuleiro, sendo que optamos por uma única lista, criar as restrições de só poder aparecer cada elemento da chave uma vez por linha/coluna e, ainda, obrigar a que a ordem da chave fosse correspondida no caracol. Daí partimos para as componentes de tornar o tamanho do tabuleiro variável, a chave variável e, por fim, incluímos o calculo do tempo de processamento do programa.

O presente relatório encontra-se estruturado segundo os seguintes tópicos:

Magic Snail, onde desenvolvemos e descrevemos o problema em questão;

Abordagem, onde descrevemos a modelação do mesmo através da enumeração das variáveis de decisão e respetivos domínios, restrições, implementação, entre outros;

Visualização da Solução, onde explicamos os predicados que permitem visualizar a solução;

Resultados, onde demonstramos exemplos de aplicação;

Conclusões, onde refletimos sobre algumas considerações face ao fim da implementação do projeto.

3 Magic Snail

O jogo Magic Snail trata-se de um problema de decisão. É jogado num tabuleiro quadrado em que as células são lidas como se este se tratasse da carapaça de um caracol, ou seja, a leitura é efetuada da cabeça do caracol para o centro.

O objetivo é que os jogadores preencham as células de forma a que, quando lidas, se obtenha sempre a chave de jogo. Os carateres da chave não se podem repetir na mesma linha ou coluna. No caso exemplificado, a chave é ABC e o tabuleiro é de 5 por 5.

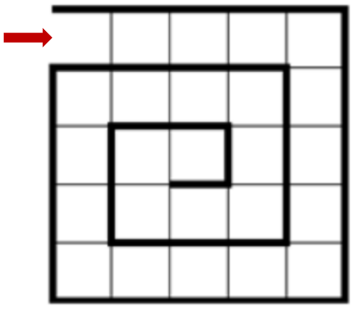

Figure 1: Tabuleiro de 5x5 vazio.

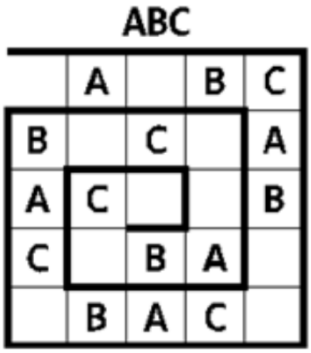

Figure 2: Solução possível do tabuleiro da figure 1 com a chave ABC

4 Abordagem

Representamos o tabuleiro como uma lista de n carateres (por exemplo, 25 carateres quando o tabuleiro é de 5x5), onde cada elemento corresponde a uma célula.

4.1 Variáveis de Decisão

Dada esta representação, existem as seguintes variáveis:

BoardLength que corresponde ao tamanho do lado do tabuleiro.

KeyLength que dá o tamanho da chave inserida pelo utilizador e que deve ser inferior ao tamanho do lado do tabuleiro.

List que é uma lista de *BoardLength*BoardLength* carateres que podem variar entre 0 e *KeyLength* e corresponde à solução do problema.

4.2 Restrições

No contexto deste problema, existem 3 restrições:

A existência da chave, independentemente da ordem dos carateres, apenas uma vez por coluna.

Ao longo de uma coluna, os elementos da chave só devem aparecer uma vez cada um.

```
oncePerColumn([],_,_):-!.
oncePerColumn(List,BoardLength, KeyLength):- oncePerColumn(List,BoardLength, KeyLength, 1).
oncePerColumn(_,BoardLength,_, N):-
  N > BoardLength,!.
oncePerColumn(List, BoardLength, KeyLength, N1):-
  Next is N1+1,
  oncePerColumn(List, BoardLength, KeyLength, Next),
  takeColumnElemsFromList(N1, BoardLength, List, Column),
  oncePerRow(Column, BoardLength, KeyLength).
```

A existência da chave, independentemente da ordem dos carateres, apenas uma vez por linha.

Ao longo de uma linha, os elementos da chave não devem aparecer repetidos.

```
oncePerRow([],_,_):-!.
oncePerRow(List, BoardLength, KeyLength):-
  takeFirstNElemsFromList(BoardLength, List, FirstElems),
  append(FirstElems,Rest,List),
  oncePerRow(Rest,BoardLength, KeyLength),
  fillListIndexEqualsValue(KeyLength,Values),
  countEqualsList(Values, FirstElems, 1).
```

A ordem dos elementos dispostos na solução, excetuando as células vazias, deve respeitar a ordem da chave.

Percorrendo o tabuleiro no sentido do caracol (ver figure 1), os carateres devem aparecer pela ordem indicada na chave. Entre estes carateres pode haver células vazias e estas não afetam a ordem.

```
magicSnailRule(List, BoardLength, KeyLength):-
  magicSnailRoute(BoardLength, Route),
  listToMagicSnail(List,Route, BoardLength, MagicSnail),
  orderedListWithCiclesIgnoring0s(MagicSnail, BoardLength, KeyLength).
```

5 Visualização da Solução

Todos os predicados responsáveis pela visualização da solução na consola, encontram-se implementados em Interface.pl.

Existem 3 predicados principais responsáveis pela estrutura do tabuleiro: verticalFrontierBoard e horizontalFrontierBoard, que tratam dos limites verticais e horizontais respetivamente, e buildBoard, responsável pela construção do tabuleiro final. Ainda, existe também a tradução dos ee (valor da célula vazia), he (valor dos limites horizontais que "não existem"), hf (valor dos limites horizontais que "existem"), ve (valor dos limites verticais que "não existem"), vf (valor dos limites verticais que "existem") e, por fim, dos valores da solução (números inteiros de valor compreendido entre 0 e *KeyLength* são convertidos nos carateres da chave). Quando na lista da solução aparece 0, traduz-se para um espaço vazio e quando aparece

valores de 1 a *KeyLength* traduz-se para o respetivo carater da chave (por exemplo, para a chave ABC, 1 corresponde a A, 2 a B e 3 a C.

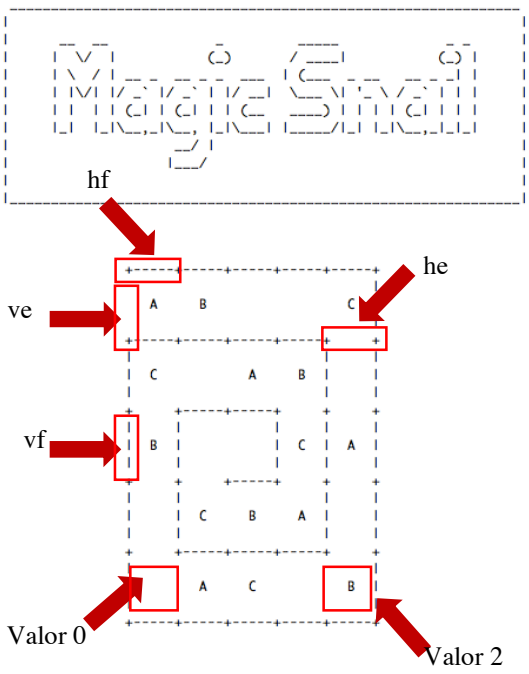

Figure 3: Chave ABC.

6 Resultados

De forma a avaliar a solução obtida, recorremos ao tempo que demora a encontrar uma solução para o problema, através do predicado statistics da biblioteca timeout.

Efetuados alguns testes para verificar a eficiência da nossa solução, obtemos os seguintes resultados:

Lado do Tabuleiro	Tamanho da Chave	Tempo (ms)
1	1	0
2	1	0
2	2	0
3	1	0
3	2	0
4	1	0
4	2	10
5	1	0
5	2	20
5	3	320
6	1	0
6	2	170
7	1	0
7	2	740
7	3	5880

Concluímos que, mantendo o tamanho do tabuleiro, se este for de pequenas dimensões, demora 0 ms a encontrar uma solução independentemente do tamanho da chave. Caso o tabuleiro seja de dimensões iguais

ou superiores a 4, conforme o tamanho da chave aumenta, o tempo de execução também vai aumentando substancialmente. E quanto maior o tabuleiro, maior o aumento para chaves do mesmo tamanho.

7 Conclusões

Terminado o desenvolvimento do projeto, concluímos que Prolog com restrições pode ser bastante útil para a resolução de certos problemas, reduzindo sensivelmente a complexidade dos mesmos. Ainda que, em alguns casos, o processamento de soluções leve algum tempo.

O jogo escolhido por nós era de reduzida complexidade teórica (só tinha 3 restrições) pelo que foi relativamente fácil percebê-lo. As maiores dificuldades terão sido permitir a existência dos espaços "vazios" entre as letras da chave no tabuleiro, processar uma chave com número variável de carateres e forçar a que a ordem dos carateres respeitasse o caracol.

É importante ressaltar que optamos por não permitir a escolha de tabuleiros retangulares, dado que não achamos que fizesse sentido, pois quebraria a dinâmica característica do jogo que é o tabuleiro ser uma carapaça de caracol.

De forma geral, pensamos que realizamos com sucesso aquilo a que nos propusemos e, com mais tempo, a única coisa que trabalharíamos seria tentar reduzir o tempo que leva à obtenção de uma solução.

8 Bibliografia

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

9 Apêndice

9.1 main.pl

```
:- include('Interface.pl').
:- include('logic.pl').
:- include('utils.pl').
:- use_module(library(lists)).
:- use_module(library(system)).
:- use_module(library(timeout)).

% --- START ---
% start(+Key, +BoardLength) -- Starts the execution of MagicSnail
start(Key, BoardLength):-
    clearScreen,
    titleFrame,
    atom_chars(Key,KeyList),
    length(KeyList, KeyLength),
    statistics(walltime,StartTime),
    magicSnail(IndexList,BoardLength,KeyLength),
    statistics(walltime,EndTime),
    statisticsMagicRule(StartTime, EndTime),
    integerToAtomUsingKey(IndexList, AtomList, KeyList),
    listToMatrix(AtomList, BoardLength, Matrix),
    buildBoard(Matrix, BoardLength, Board),
    displayBoard(Board),
    anotherSolution.

% statisticsMagicRule(+Runtime) -- Show runtime of MagicSnail
statisticsMagicRule(StartTime, EndTime):-
    nth1(1,StartTime, StartTimeMs),
    nth1(1,EndTime, EndTimeMs),
    RunTime is EndTimeMs - StartTimeMs,
    write('Runtime: '), write(RunTime), write(' ms.'),
    spacing(2).
```

9.2 logic.pl

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

magicSnail(List, BoardLength, KeyLength):-
    ListLength is BoardLength*BoardLength,
    length(List, ListLength),
    domain(List, 0, KeyLength),
```

```
    oncePerRow(List, BoardLength, KeyLength),
    oncePerColumn(List, BoardLength, KeyLength),
    magicSnailRule(List, BoardLength, KeyLength),
    labeling([], List).

oncePerRow([],_,_):-!.
oncePerRow(List, BoardLength, KeyLength):-
    takeFirstNElemsFromList(BoardLength, List, FirstElems),
    append(FirstElems,Rest,List),
    oncePerRow(Rest,BoardLength, KeyLength),
    fillListIndexEqualsValue(KeyLength,Values),
    countEqualsList(Values, FirstElems, 1).

oncePerColumn([],_,_):-!.
oncePerColumn(List,BoardLength, KeyLength):- oncePerColumn(List,BoardLength,
KeyLength, 1).
oncePerColumn(_,BoardLength,_, N):-
    N > BoardLength,!.
oncePerColumn(List, BoardLength, KeyLength, N1):-
    Next is N1+1,
    oncePerColumn(List, BoardLength, KeyLength, Next),
    takeColumnElemsFromList(N1, BoardLength, List, Column),
    oncePerRow(Column, BoardLength, KeyLength).

magicSnailRoute([d,d,d,d,s,s,s,s,a,a,a,a,w,w,w,d,d,d,s,s,a,a,w,d]).
magicSnailRoute(1,[]):-!.
magicSnailRoute(2,[d,s,a]):-!.
magicSnailRoute(N,Route):-
    N2 is N-2,
    magicSnailRoute(N2, RouteRest),
    N1 is N-1,
    fillListWithValue(D, d, N1),
    fillListWithValue(S, s, N1),
    fillListWithValue(A, a, N1),
    fillListWithValue(W, w, N2),
    append(D, S, DS),
    append(DS, A, DSA),
    append(DSA, W, DSAW),
    append(DSAW, [d], RouteN),
    append(RouteN, RouteRest, Route).

magicSnailRule(List, BoardLength, KeyLength):-
    magicSnailRoute(BoardLength, Route),
    listToMagicSnail(List,Route, BoardLength,MagicSnail),
    orderedListWithCiclesIgnoring0s(MagicSnail,BoardLength, KeyLength).
```

```
orderedListWithCiclesIgnoring0s(List,BoardLength, KeyLength):-
    orderedListWithCiclesIgnoring0s(List,BoardLength, KeyLength,1,0).
orderedListWithCiclesIgnoring0s(_,BoardLength,_, N,_):-
    N>BoardLength.
orderedListWithCiclesIgnoring0s(List,BoardLength, KeyLength,Occurrence,
LastOccurrencePrev):-
    fillListIndexEqualsValue(KeyLength, Values),
    nthOccurrenceListing(List, Values, Occurrence, LastOccurrencePrev,
LastOccurrenceNext),
    NextOccurrence is Occurrence+1,
    orderedListWithCiclesIgnoring0s(List,BoardLength, KeyLength,NextOccurrence,
LastOccurrenceNext).

nthOccurrenceListing(_, [], _, LastOccurrence, LastOccurrence):-!.
nthOccurrenceListing(List, [Value|Rest], Occurrence, LastOccurrencePrev,
LastOccurrenceNext):-
nthOccurrence(List, Value, Occurrence, Index),
    LastOccurrencePrev#<Index,
    nthOccurrenceListing(List, Rest, Occurrence, Index, LastOccurrenceNext).

numberOfOcurrencesUntil(_,_,Count,Count,Index,Index).
numberOfOcurrencesUntil(List, Element, Aux, Count, Index, IndexAux) :-
    List = [H|T],
    H #= Element #<=> YES,
    Aux2 #= Aux + YES,
    N_Index is IndexAux + 1,
    numberOfOcurrencesUntil(T, Element, Aux2, Count, Index, N_Index).

%O elemento Element aparece na lista L pela NthTime no index Index
nthOccurrence(List, Element, NthTime, Index) :-
  numberOfOcurrencesUntil(List,Element, 0, NthTime, Index, 0), %restrição de até aquele
index o element aparecer nthtime
   element(Index,List,Element). % restrição de naquela posição ser aquele element

%countEquals(Value, List, Count) - Count equal values on a list
countEquals(_, [], 0).
countEquals(Value, [Head | Tail], Count) :-
    countEquals(Value, Tail, Count2),
    Value #= Head #<=> Flag,
    Count #= Count2 + Flag.

countEqualsList([],_,_).
countEqualsList([Value|Rest], List, Count):-
    countEqualsList(Rest, List, Count),
    countEquals(Value,List,Count).
```

9.3 Interface.pl

```
initialBoard([
    [ee, ee, ee, ee, ee],
    [ee, ee, ee, ee, ee],
    [ee, ee, ee, ee, ee],
    [ee, ee, ee, ee, ee],
    [ee, ee, ee, ee, ee]
]).

initialBoard(N, Board):-
    fillListWithValue(Row, ee, N),
    fillListWithValue(Board,Row,N).

exampleBoard([
    [ee, 'A', ee, 'B', 'C'],
    ['B', ee, 'C', ee, 'A'],
    ['A', 'C', ee, ee, 'B'],
    ['C', ee, 'B', 'A', ee],
    [ee, 'B', 'A', 'C', ee]
]).

horizontalFrontierBoard([
    [hf, hf, hf, hf, hf],
    [hf, hf, hf, hf, he],
    [he, hf, hf, he, he],
    [he, he, hf, he, he],
    [he, hf, hf, hf, he],
    [hf, hf, hf, hf, hf]
]).

horizontalFrontierBoard(0, [[]]):- !.
horizontalFrontierBoard(1, [[hf],[hf]]):-!.
horizontalFrontierBoard(N, FrontierBoard):-
    N2 is N-2,
    horizontalFrontierBoard(N2,FrontierBoardRest),
    fillListWithValue(Top, hf, N),
    horizontalFrontierBoardAux(N, FrontierBoardRest, Center),
    fillListWithValue(Bottom, hf, N),
    append([Top], Center, TC),
    append(TC, [Bottom], FrontierBoard).

horizontalFrontierBoardAux(_,[],[]).
horizontalFrontierBoardAux(N, [Row|Rest], Center):-
    N1 is N-1,
    length([Row|Rest],N1),
```

```
    !,
    horizontalFrontierBoardAux(N, Rest, CenterRest),
    append([hf],Row,HFR),
    append(HFR,[he],NewRow),
    append([NewRow], CenterRest, Center).

horizontalFrontierBoardAux(N, [Row|Rest], Center):-
    !,
    horizontalFrontierBoardAux(N, Rest, CenterRest),
    append([he],Row,HER),
    append(HER,[he],NewRow),
    append([NewRow], CenterRest, Center).

verticalFrontierBoard([
    [ve, ve, ve, ve, ve, vf],
    [vf, ve, ve, ve, vf, vf],
    [vf, vf, ve, vf, vf, vf],
    [vf, vf, ve, ve, vf, vf],
    [vf, ve, ve, ve, ve, vf]
]).

verticalFrontierBoard(0, []):- !.
verticalFrontierBoard(1, [[ve,vf]]):-!.
verticalFrontierBoard(N, FrontierBoard):-
    N2 is N-2,
    verticalFrontierBoard(N2, FrontierBoardRest),
    fillListWithValue(ATop, ve, N),
    append(ATop, [vf], Top),
    verticalFrontierBoardAux(N, FrontierBoardRest, Center),
    N1 is N-1,
    fillListWithValue(CBottom,ve,N1),
    append([vf], CBottom, LBottom),
    append(LBottom, [vf], Bottom),
    append([Top], Center, TC),
    append(TC, [Bottom], FrontierBoard).

verticalFrontierBoardAux(_,[],[]).
verticalFrontierBoardAux(N, [Row|Rest], Center):-
    verticalFrontierBoardAux(N, Rest, CenterRest),
    append([vf],Row,VFR),
    append(VFR,[vf],NewRow),
    append([NewRow], CenterRest, Center).

% -- Logo --
titleFrame :-
        write(' ____________________________________________________________________________'),nl,
```

```
        write('|                                                    |'),nl,
        write('|    __  __           _       _____          _ _     |'), nl,
        write('|    |  \\/  |         (_)     / ____|        (_) |    |'), nl,
        write('|    | \\  / | __ _  __ _ _ __ | (___  _ __   __ _| |    |'), nl,
        write('|    | |\\/| |/ _` |/ _` | |/ __|  \\___ \\| \'_ \\ / _` | | |    |'), nl,
        write('|    | |  | | (_| | (_| | | (__   ____) | | | | (_| | | |    |'), nl,
        write('|    |_|  |_|\\__,_|\\__, |_|\\___| |_____/|_| |_|\\__,_|_|_|    |'), nl,
        write('|                 __/ |                                |'), nl,
        write('|                |___/                                 |'), nl,
        write('|                                                    |'),nl,
        write('|____________________________________________________|'),nl,
        spacing(3).

    buildBoard(Matrix, BoardLength, Board):-
        horizontalFrontierBoard(BoardLength, HF),
        verticalFrontierBoard(BoardLength, VF),
        buildBoard(Matrix, HF, VF, BoardLength, Board).
    buildBoard([], [RowHF], [], _, [Result]):-
        horizontalFrontierParser(RowHF, Result).
    buildBoard([RowElems|RestElems], [RowHF|RestHF], [RowVF|RestVF], BoardLength,
Result):-buildBoard(RestElems, RestHF, RestVF, BoardLength, RestResult),
        fillListWithValue(NEmpty, ee, BoardLength),
        horizontalFrontierParser(RowHF, RowHead),
        verticalFrontierParser(RowVF, NEmpty, RowContentTop),
        verticalFrontierParser(RowVF, RowElems, RowContent),
        verticalFrontierParser(RowVF, NEmpty, RowContentBottom),
        Result = [RowHead, RowContentTop, RowContent, RowContentBottom|RestResult].

    displayBoard([]).
    displayBoard([Row|Rest]):-
        write('    '),
        displayRow(Row), nl,
        displayBoard(Rest).

    horizontalFrontierParser([], ['+']).
    horizontalFrontierParser([R1|Rest], Result):-
        horizontalFrontierParser(Rest,RecResult),
        translate(R1, Ascii),
        Result = [Ascii|RecResult].

    verticalFrontierParser([R1], [], Result):-
        translate(R1, Ascii),
        Result = [Ascii].

    verticalFrontierParser([R1|Rest], [Elem|RemainingRow], Result):-
        verticalFrontierParser(Rest, RemainingRow, RecResult),
```

```
    translate(R1, Ascii),
    translate(Elem, TElem),
    Result = [Ascii, TElem  | RecResult].

translate(ee, '     '):-!.
translate(he, '+     '):-!.
translate(hf, '+-----'):-!.
translate(ve, ' '):-!.
translate(vf, '|'):-!.
translate(X, Res):-
    atom_concat('  ', X, X1),
    atom_concat(X1, '  ', Res).

displayRow([]).
displayRow([Elem | Rest]):-
    write(Elem),
    displayRow(Rest).

% -- Visualization --
spacing(0):-!.
spacing(N):-
    N1 is N-1,
    spacing(N1),
    nl.

clearScreen :- spacing(50), !.

getCharThenEnter(X) :-
        get_char(X),
        get_char(_), !.

anotherSolution:-
    spacing(2),
    write('Another Solution? (y/n) '),
    spacing(2),
    getCharThenEnter(X),
    spacing(3),
    switch(X,[y:fail, n:true]).
```

9.4 utils.pl

```
fillListWithValue([], _, 0):-!.
fillListWithValue([Value | Rest], Value, N) :- N1 is N-1, fillListWithValue(Rest, Value, N1).

fillListIndexEqualsValue(0, []):-!.
```

```
fillListIndexEqualsValue(N, List):-
   N1 is N-1,
   fillListIndexEqualsValue(N1,ListRest),
   append(ListRest,[N],List).

takeFirstNElemsFromList(N, List, Partial) :-
   takeFirstNElemsFromList(N, List, Partial,0).
takeFirstNElemsFromList(Count, _, [],Count).
takeFirstNElemsFromList(N, [Value|Rest], Partial,Count) :-
   Count1 is Count+1,
   takeFirstNElemsFromList(N, Rest, Partial1,Count1),
   append([Value], Partial1, Partial).

takeColumnElemsFromList(_,_,[],[]):-!.
takeColumnElemsFromList(NColumn, BoardLength, List, Result):-
   takeFirstNElemsFromList(BoardLength, List, FirstElems),
   append(FirstElems,Rest,List),
   takeColumnElemsFromList(NColumn, BoardLength, Rest, Result1),
   nth1(NColumn, FirstElems, Value),
   append([Value], Result1, Result).

listToMatrix([], _, []).
listToMatrix(List, Size, [Row|Matrix]):-
  listToMatrixRow(List, Size, Row, Tail),
  listToMatrix(Tail, Size, Matrix).

listToMatrixRow(Tail, 0, [], Tail).
listToMatrixRow([Item|List], Size, [Item|Row], Tail):-
  NSize is Size-1,
  listToMatrixRow(List, NSize, Row, Tail).

% integerToAtomUsingKey(+Before, -After, +Key)
integerToAtomUsingKey([], [], _).
integerToAtomUsingKey([Elem|Rest], After, Key):-
   integerToAtomUsingKey(Rest, AfterRest, Key),
   ifelse(nth1(Elem, Key, Atom),
      After = [Atom|AfterRest],
      After = [ee|AfterRest]).

removeElementFromList(_, [], []) :- !.
removeElementFromList(Elem, [Elem|Rest], Result) :-
   !,
   removeElementFromList(Elem, Rest, Result).
removeElementFromList(Elem, [NotElem|Rest], Result) :-
   !,
   removeElementFromList(Elem, Rest, ResultRest),
```

```
    append([NotElem], ResultRest, Result).

% nthOccurrenceInList(+List, ?Element, ?Index, ?Occurences)
nthOccurrenceInList(List, Element, Index, Occurences):-
    nthOccurrenceInList(List, Element, Index, 0, Occurences).
nthOccurrenceInList(_, _, 0, Occurences, Occurences) :- !.
nthOccurrenceInList([Element|Tail], Element, Index, Count, Occurences):-
    Count2 is Count+1,
    !,nthOccurrenceInList(Tail, Element, Index1, Count2, Occurences),
    Index is Index1+1.
nthOccurrenceInList([_|Tail], Element, Index, Count, Occurences):-
    !,nthOccurrenceInList(Tail, Element, Index1, Count, Occurences),
    Index is Index1+1.

% --- Magic Snail process ---
listToMagicSnail(List, Route, BoardLength, MagicSnail):-
    listToMagicSnail(List, Route, BoardLength, MagicSnail, 0,0).
listToMagicSnail(List, [], BoardLength, MagicSnail ,Row,Column):-
    Index is Row*BoardLength + Column,
    nth0(Index, List, Elem),
    MagicSnail = [Elem].
listToMagicSnail(List, [Dir|RestRoute], BoardLength, MagicSnail, Row,Column):-
    switch(Dir, [
        d:(NColumn is Column+1,
            NRow is Row),
        a:(NColumn is Column-1,
            NRow is Row),
        s:(NColumn is Column,
            NRow is Row+1),
        w:(NColumn is Column,
            NRow is Row-1)]),
    listToMagicSnail(List, RestRoute, BoardLength, MagicSnailRest,NRow,NColumn),
    Index is Row*BoardLength + Column,
    nth0(Index, List, Elem),
    MagicSnail = [Elem | MagicSnailRest].

% --- operators ---
ifelse(Condition, If, _Else) :- Condition, !, If.
ifelse(_, _, Else) :- Else.

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

switch(X, [Case:Then|Cases]) :-
    ( X=Case ->call(Then);
        switch(X, Cases)).
```