RELATÓRIO DE DESENVOLVIMENTO PRÁTICO -> TRABALHO III

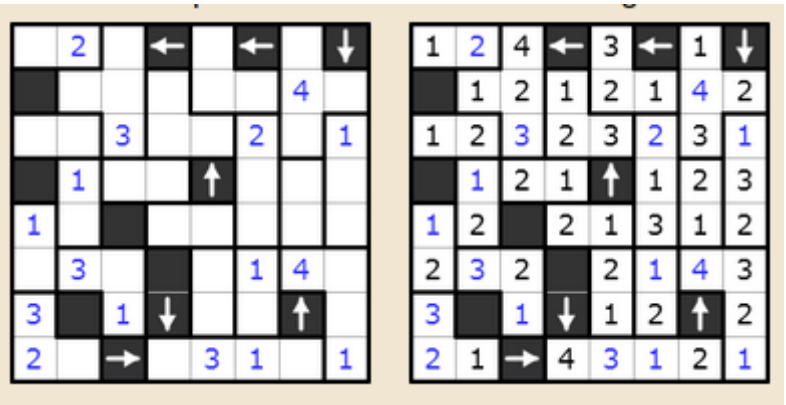
INE5416 - Paradigmas de Programação

Puzzle: Makaro (マカロ)

Alunos: Raquel Behrens, Tainá da Cruz, Vitor Marzarotto

1. Análise do Problema

Makaro (マカロ) é um quebra-cabeça lógico publicado pela primeira vez na revista japonesa de quebra-cabeças Nikoli.



Existem 3 regras para resolver esse puzzle:

1. Inserir um número em cada célula branca do diagrama para que cada região de N células contenha todos os números de 1~N exatamente uma vez.
2. Os mesmos números não devem ser ortogonalmente adjacentes.
3. Uma seta em uma célula preta aponta para a célula adjacente ortogonalmente com o número absolutamente mais alto.

2. Entrada do Usuário

Para resolver uma matriz makaro com o programa é preciso entender como a matriz que representa o tabuleiro é definida no código. Além disso, a entrada da matriz do puzzle é atualizada diretamente no código.

A matriz *makaro* está localizada no arquivo ‘main.pl’. Ela é formada por uma lista de listas, as quais representam as linhas. Cada linha é composta por n células. A célula é uma estrutura que possui quatro valores inteiros:

- *1º valor:* Representa a linha;
- *2º valor:* Representa a coluna;
- *3º valor:* Representa a região da célula;

- *4º valor:* Representa o valor da célula.

Se a célula for uma célula com valor fixo ou ainda não preenchida, seu 3º valor (região) terá que ser maior que 0. Se seu valor for fixo, seu 4º valor terá que ter o valor original, e se ainda não estiver preenchido, terá que ser igual a _.

Se a célula for uma célula que representa uma flecha ou uma célula preta (sem nada e impreenchível), seu 3º valor (região) terá que ser igual a 0. Para essas situações, o 4º valor deverá ser preenchido da seguinte forma:

- 0 - caso for uma célula preta;
- 1 - caso a flecha estiver apontando para a direita;
- 2 - caso a flecha estiver apontando para baixo;
- 3 - caso a flecha estiver apontando para a esquerda;
- 4 - caso a flecha estiver apontando para cima.

Para executar o programa, deve-se ter *swipl* instalado no computador, compilar o arquivo main com:

- $ swipl -o main -c main.pl

  Executar o arquivo compilado com:
- $ ./main

  Executar a função para solucionar o makaro com:
- $ makaro(2,Rows), solucao(Rows).

3. Estratégia e Solução Aplicada

A estratégia aplicada se baseou na técnica de programação de “tentativa e erro” (*backtracking*), realizada automaticamente utilizando a linguagem “Prolog”. Na função **solucao** (main,pl), é chamado a função **completa** para todas as células da matriz.

A função **completa** (*main.lp*) do código funciona preenchendo um valor em cada célula, e chama a função **verifica** (main.lp) para verificar se o valor preenchido está de acordo com as regras do jogo.

A função **verifica**, chama a função **naoHalgualNaRegiao** (pruning,pl), que checa os números já preenchidos de cada região, e retorna falso caso encontre um número já preenchido igual ao que acabou de ser preenchido na função **completa**.

Após isso, chama a função **verificarAdjacencias**, que verifica se o número preenchido é igual a algum número já preenchido em alguma de suas adjacências. E, por fim, chama a função **verificarFlechasMatriz**, que verifica todas as células que possuem flechas, e checa se a flecha aponta para o maior número de suas adjacências.

- *main* - possui a função **makaro** (onde é definida a matriz makaro), e as funções: **solução**, **completa** e **verifica**:

```
main:- makaro(2,Rows),solucao(Rows).


makaro(2, M) :-
    M = [[cell(1,1,0,1),  cell(1,2,2,_),   cell(1,3,2,4),  cell(1,4,0,3),  cell(1,5,3,_),  cell(1,6,4,_),  cell(1,7,4,4),  cell(1,8,4,1)],
         [cell(2,1,1,_),  cell(2,2,2,_),   cell(2,3,2,_),  cell(2,4,3,_),  cell(2,5,3,1),  cell(2,6,4,2),  cell(2,7,6,1),  cell(2,8,0,2)],
         [cell(3,1,1,_),  cell(3,2,0,1),   cell(3,3,5,4),  cell(3,4,5,2),  cell(3,5,0,0),  cell(3,6,6,_),  cell(3,7,6,2),  cell(3,8,6,4)],
         [cell(4,1,16,2), cell(4,2,16,_),  cell(4,3,0,4),  cell(4,4,5,_),  cell(4,5,5,3),  cell(4,6,9,_),  cell(4,7,0,1),  cell(4,8,6,5)],
         [cell(5,1,0,1),  cell(5,2,7,_),   cell(5,3,7,_),  cell(5,4,8,_),  cell(5,5,8,_),  cell(5,6,9,_),  cell(5,7,9,_),  cell(5,8,10,_)],
         [cell(6,1,17,_), cell(6,2,14,_),  cell(6,3,7,_),  cell(6,4,0,4),  cell(6,5,8,2),  cell(6,6,8,1),  cell(6,7,0,2),  cell(6,8,10,2)],
         [cell(7,1,0,1),  cell(7,2,14,5),  cell(7,3,13,_), cell(7,4,13,2), cell(7,5,12,_), cell(7,6,0,1),  cell(7,7,10,3), cell(7,8,10,_)],
         [cell(8,1,14,3), cell(8,2,14,_),  cell(8,3,14,4), cell(8,4,13,_), cell(8,5,12,_), cell(8,6,11,1), cell(8,7,11,_), cell(8,8,0,4)]].



solucao(TabuleiroSolucao) :-

    TabuleiroSolucao = [[X11, X12, X13, X14, X15, X16, X17, X18],
                        [X21, X22, X23, X24, X25, X26, X27, X28],
                        [X31, X32, X33, X34, X35, X36, X37, X38],
                        [X41, X42, X43, X44, X45, X46, X47, X48],
                        [X51, X52, X53, X54, X55, X56, X57, X58],
                        [X61, X62, X63, X64, X65, X66, X67, X68],
                        [X71, X72, X73, X74, X75, X76, X77, X78],
                        [X81, X82, X83, X84, X85, X86, X87, X88]
                       ],

    %linha1
    completa(X11, TabuleiroSolucao),
    completa(X12, TabuleiroSolucao),
    completa(X13, TabuleiroSolucao),
    completa(X14, TabuleiroSolucao),
    completa(X15, TabuleiroSolucao),
```

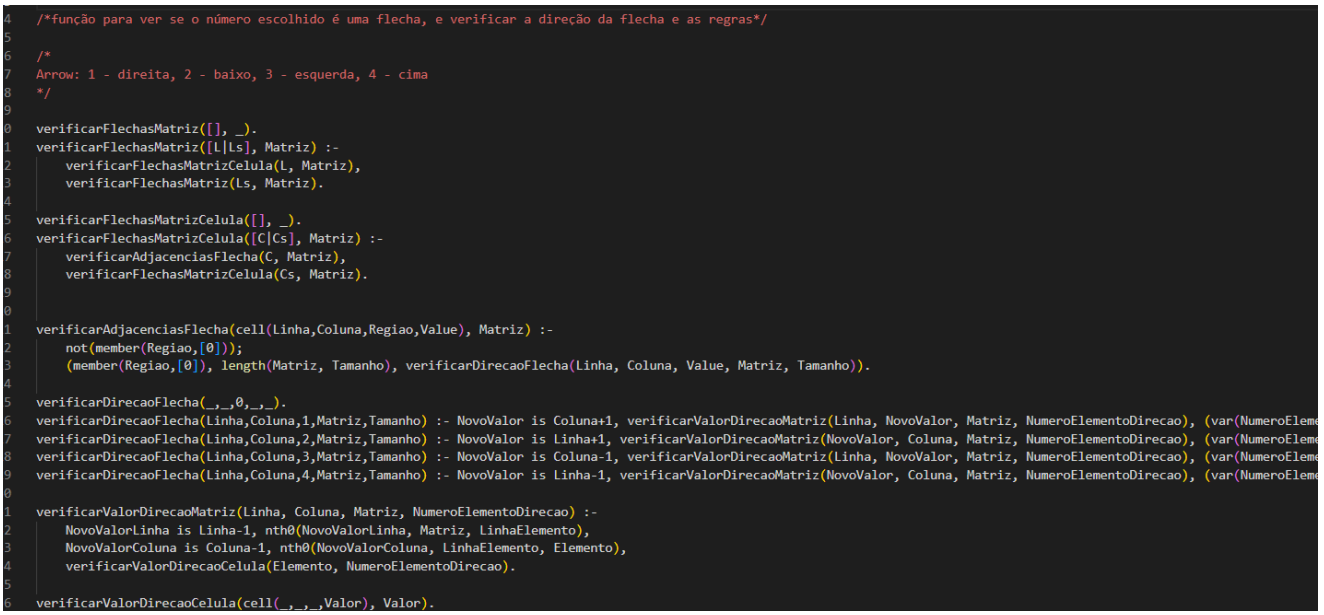
```
116
117
118   /*função que completa o número na célula*/
119
120   completa(cell(_,_,0,_), _).
121   completa(cell(Linha,Coluna,Region,Value), Matriz) :-
122       acharQuantidadeElementosRegiao(Region, Matriz, QuantidadeElementosRegiao),
123       my_in(Value, 1, QuantidadeElementosRegiao),
124       verifica(Linha,Coluna,Value, Region, Matriz).
125
126   verifica(Linha,Coluna,Value,Region,Matriz) :-
127       naoHaIgualNaRegiao(Linha,Coluna,Value, Region, Matriz),
128       verificarAdjacencias(Matriz),
129       verificarFlechasMatriz(Matriz, Matriz).
130
131   my_in(Value, Maior, Maior) :- Value is Maior.
132   my_in(Value, Menor, Maior) :- Value is Menor ; my_in(Value, (Menor+1), Maior).
133
```

- *Adjacency* - Agrupa as funções que verificam as adjacências do número escolhido, e retornam falso caso alguma adjacência tiver o seu valor igual a este:

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

/*fazer função para ver se o número escolhido está ao lado de um número igual*/
verificarAdjacencias(TabuleiroSolucao) :-

    TabuleiroSolucao = [[X11, X12, X13, X14, X15, X16, X17, X18],
                        [X21, X22, X23, X24, X25, X26, X27, X28],
                        [X31, X32, X33, X34, X35, X36, X37, X38],
                        [X41, X42, X43, X44, X45, X46, X47, X48],
                        [X51, X52, X53, X54, X55, X56, X57, X58],
                        [X61, X62, X63, X64, X65, X66, X67, X68],
                        [X71, X72, X73, X74, X75, X76, X77, X78],
                        [X81, X82, X83, X84, X85, X86, X87, X88]
                       ],

    %cantos
    naoHaIgualAoLadoCanto(X11, X12, X21),
    naoHaIgualAoLadoCanto(X81, X82, X71),
    naoHaIgualAoLadoCanto(X18, X17, X28),
    naoHaIgualAoLadoCanto(X88, X78, X87),

    %linhaEsquerda
    naoHaIgualAoLadoLinha(X21, X11, X31, X22),
    naoHaIgualAoLadoLinha(X31, X21, X41, X32),
    naoHaIgualAoLadoLinha(X41, X31, X51, X42),
    naoHaIgualAoLadoLinha(X51, X41, X61, X52),
    naoHaIgualAoLadoLinha(X61, X51, X71, X62),
    naoHaIgualAoLadoLinha(X71, X61, X81, X72),

    %linhaDireita
    naoHaIgualAoLadoLinha(X28, X18, X38, X27),
```

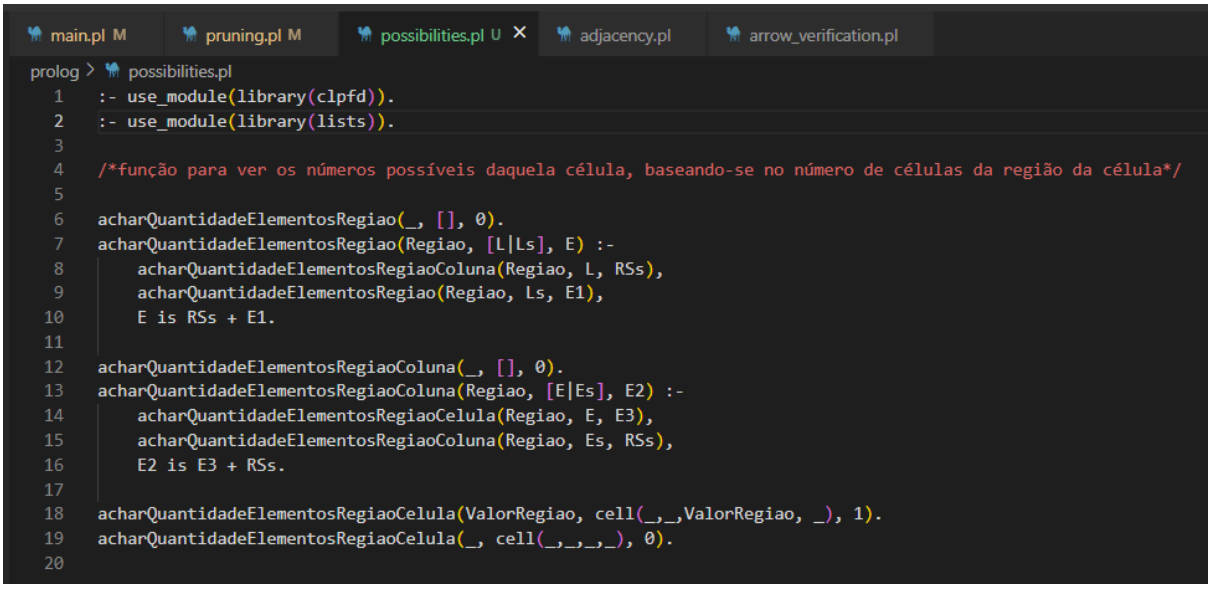
```
naoHaIgualAoLadoCanto(cell(_,_,Region1, Value1), cell(_,_,Region2, Value2), cell(_,_,Region3, Value3)) :-
                                                                    member(Region1, [0]) ;
                                                                    (not(member(Region1, [0])) ,
                                                                        (var(Value1) ;
                                                                         ((member(Region2, [0]) ;
                                                                          (not(member(Region2, [0])) ,
                                                                              (var(Value2) ;
                                                                               not(member(Value1, [Value2]))))) ,
                                                                         (member(Region3, [0]) ;
                                                                          (not(member(Region3, [0])) ,
                                                                              (var(Value3) ;
                                                                               not(member(Value1, [Value3])))))))).


naoHaIgualAoLadoLinha(cell(_,_,Region1, Value1), cell(_,_,Region2, Value2), cell(_,_,Region3, Value3), cell(_,_,Region4, Value4)) :-
                                                                    member(Region1, [0]) ;
                                                                    (not(member(Region1, [0])) ,
                                                                        (var(Value1) ;
                                                                         ((member(Region2, [0]) ;
                                                                          (not(member(Region2, [0])) ,
                                                                              (var(Value2) ;
                                                                               not(member(Value1, [Value2]))))) ,
                                                                          ((member(Region3, [0]) ;
                                                                           (not(member(Region3, [0])) ,
                                                                               (var(Value3) ;
                                                                                not(member(Value1, [Value3]))))),
                                                                           (member(Region4, [0]) ;
                                                                           (not(member(Region4, [0])) ,
                                                                               (var(Value4) ;
                                                                                not(member(Value1, [Value4])))))
                                                                          )))).
```

- *Arrow_verification* - Agrupa as funções que verificam as adjacências de cada flecha da matriz, e retornam falso caso a flecha não esteja apontando para o maior número:



```
/*função para ver se o número escolhido é uma flecha, e verificar a direção da flecha e as regras*/

/*
Arrow: 1 - direita, 2 - baixo, 3 - esquerda, 4 - cima
*/

verificarFlechasMatriz([], _).
verificarFlechasMatriz([L|Ls], Matriz) :-
    verificarFlechasMatrizCelula(L, Matriz),
    verificarFlechasMatriz(Ls, Matriz).

verificarFlechasMatrizCelula([], _).
verificarFlechasMatrizCelula([C|Cs], Matriz) :-
    verificarAdjacenciasFlecha(C, Matriz),
    verificarFlechasMatrizCelula(Cs, Matriz).


verificarAdjacenciasFlecha(cell(Linha,Coluna,Regiao,Value), Matriz) :-
    not(member(Regiao,[0])),
    (member(Regiao,[0]), length(Matriz, Tamanho), verificarDirecaoFlecha(Linha, Coluna, Value, Matriz, Tamanho)).

verificarDirecaoFlecha(_,_,0,_,_).
verificarDirecaoFlecha(Linha,Coluna,1,Matriz,Tamanho) :- NovoValor is Coluna+1, verificarValorDirecaoMatriz(Linha, NovoValor, Matriz, NumeroElementoDirecao), (var(NumeroEleme
verificarDirecaoFlecha(Linha,Coluna,2,Matriz,Tamanho) :- NovoValor is Linha+1, verificarValorDirecaoMatriz(NovoValor, Coluna, Matriz, NumeroElementoDirecao), (var(NumeroEleme
verificarDirecaoFlecha(Linha,Coluna,3,Matriz,Tamanho) :- NovoValor is Coluna-1, verificarValorDirecaoMatriz(Linha, NovoValor, Matriz, NumeroElementoDirecao), (var(NumeroEleme
verificarDirecaoFlecha(Linha,Coluna,4,Matriz,Tamanho) :- NovoValor is Linha-1, verificarValorDirecaoMatriz(NovoValor, Coluna, Matriz, NumeroElementoDirecao), (var(NumeroEleme

verificarValorDirecaoMatriz(Linha, Coluna, Matriz, NumeroElementoDirecao) :-
    NovoValorLinha is Linha-1, nth0(NovoValorLinha, Matriz, LinhaElemento),
    NovoValorColuna is Coluna-1, nth0(NovoValorColuna, LinhaElemento, Elemento),
    verificarValorDirecaoCelula(Elemento, NumeroElementoDirecao).

verificarValorDirecaoCelula(cell(_,_,_,Valor), Valor).
```

- *Possibilities* - Agrupa as funções que verificam qual o maior número possível de uma célula:



main.pl M | pruning.pl M | possibilities.pl U | adjacency.pl | arrow_verification.pl

prolog > possibilities.pl

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

/*função para ver os números possíveis daquela célula, baseando-se no número de células da região da célula*/

acharQuantidadeElementosRegiao(_, [], 0).
acharQuantidadeElementosRegiao(Regiao, [L|Ls], E) :-
    acharQuantidadeElementosRegiaoColuna(Regiao, L, RSs),
    acharQuantidadeElementosRegiao(Regiao, Ls, E1),
    E is RSs + E1.

acharQuantidadeElementosRegiaoColuna(_, [], 0).
acharQuantidadeElementosRegiaoColuna(Regiao, [E|Es], E2) :-
    acharQuantidadeElementosRegiaoCelula(Regiao, E, E3),
    acharQuantidadeElementosRegiaoColuna(Regiao, Es, RSs),
    E2 is E3 + RSs.

acharQuantidadeElementosRegiaoCelula(ValorRegiao, cell(_,_,ValorRegiao, _), 1).
acharQuantidadeElementosRegiaoCelula(_, cell(_,_,_,_), 0).

```

- Prunning - Agrupa as funções que verificam se o número escolhido já não pertence à região daquela célula:

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


/*terminar função para ver se o número escolhido já não pertence à região daquela célula*/

naoHaIgualNaRegiao(Linha, Coluna, Value, Regiao, Matriz) :-
    acharElementosRegiao(Linha, Coluna, Regiao, Matriz, ElementosRegiao),
    not(member(Value,ElementosRegiao)).

acharElementosRegiao(_, _, _, [], []).
acharElementosRegiao(Linha, Coluna, Regiao, [L|Ls], E) :-
    acharElementosRegiaoColuna(Linha, Coluna, Regiao, L, RSs),
    acharElementosRegiao(Linha, Coluna, Regiao, Ls, E1),
    append(RSs, E1, E).

acharElementosRegiaoColuna(_,_,_, [], []).
acharElementosRegiaoColuna(Linha, Coluna, Regiao, [cell(Linha,Coluna,Regiao,_)|Es], E2) :-
    acharElementosRegiaoColuna(Linha, Coluna, Regiao, Es, E2).
acharElementosRegiaoColuna(Linha, Coluna, Regiao, [cell(_,_,Regiao,Elemento)|Es], [Elemento|E2]) :-
    not(var(Elemento)), acharElementosRegiaoColuna(Linha, Coluna, Regiao, Es, E2).
acharElementosRegiaoColuna(Linha, Coluna, Regiao, [cell(_,_,Regiao,Elemento)|Es], E2) :-
    var(Elemento), acharElementosRegiaoColuna(Linha, Coluna, Regiao, Es, E2).
acharElementosRegiaoColuna(Linha, Coluna, Regiao, [cell(_,_,RegiaoCelula,_)|Es], E2) :-
    not(member(RegiaoCelula, [Regiao])), acharElementosRegiaoColuna(Linha, Coluna, Regiao, Es, E2).

```

4. Vantagens e Desvantagens entre Haskell e Lisp

A **vantagem** observada na linguagem PROLOG é a facilidade em criar regras para encontrar uma solução para o problema, e o backtracking ser realizado automaticamente caso uma das regras seja quebrada, diferentemente dos programas construídos em HASKELL e LISP. Por conta disso, não foi preciso construir uma função para realizar o backtracking na solução do problema.

A **desvantagem** observada é a dificuldade em *debugar* o código e encontrar problemas existentes. Caso o código seja muito complexo, o processo de encontrar algum bug e resolvê-lo é extremamente demorado e trabalhoso, já que, caso alguma regra seja quebrada, o programa realiza backtracking automaticamente, podendo entrar em algum *if* construído de maneira incorreta de alguma função, e retornar verdadeiro erroneamente.

Dessa forma, o grupo achou mais fácil construir um programa resolvedor de Makaro em LISP e HASKELL, do que em PROLOG.

5. Organização do Grupo

O grupo encontrou-se por chamada de voz para resolver o trabalho.

6. Dificuldades Encontradas

As principais dificuldades encontradas foram:

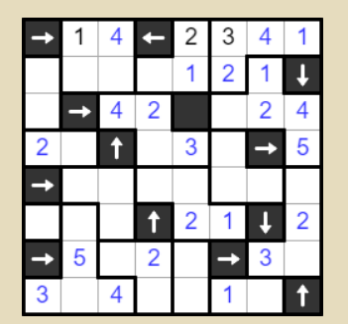
1. Como as possibilidades de valores de cada célula variam de acordo com o número de células que cada região possui, o grupo encontrou dificuldade em construir corretamente a função que escolhe o número de uma célula:

```
132   my_in(Value, Maior, Maior) :- Value is Maior.
133   my_in(Value, Menor, Maior) :- Value is Menor ; my_in(Value, (Menor+1), Maior).
134
```

Toda vez que o programa escolhe o valor errado, e volta para essa função para escolher o próximo valor, o programa escolhe valores cada vez maiores e não respeita o máximo valor possível (definido como *Maior* no código).

O grupo não conseguiu resolver esse problema.

2. O programa não realiza o backtracking como o esperado. Para ilustrar melhor, será descrito um exemplo do problema:
    - Nas primeiras execuções do programa, os números escolhidos para cada célula são os seguintes (é possível observar que o número 1, na linha 1, coluna 2, é o número incorreto da posição):



- Quando o programa vai escolher o número da linha 2, coluna 1, ele não consegue escolher nenhum número, ou seja, todas as verificações dão errado.

Assim, o programa fica em um loop infinito, e nunca consegue decidir o número dessa célula, já que todas as verificações dão falso.

Mesmo todas as verificações dando falso, o programa não volta para a linha 1, coluna 2, para escolher outro número.

O grupo não conseguiu resolver esse problema.