Lógica para Programação

Projecto

2024-2025

Quando estrelas dão belos predicados

Conteúdo

1 Introdução

Risca o que não interessa e/ou acrescenta uma experiência onde apareça a palavra “estrela”. Acabaste de:

- ver um filme/episódio qualquer da saga da “Guerra das Estrelas”,
- ouvir a música do Rui Veloso "Não há estrelas no céu",
- ler "A culpa é das estrelas",
- encontrar uma estrela do mar na praia,
- plantar uma Estrela-do-Egipto,
- ir à Serra da Estrela,
- comer um ovo estrelado,
- _____.

Esta experiência foi tão boa que fazes uma pesquisa com a palavra “estrela”. Aparece-te uma *app* chamada“Star Battle”. Com curiosidade, instalas a *app* e achas incríveis os seus desafios. Decides então implementar, em Prolog, um programa que resolva esses desafios; talvez não os resolvas todos, mas vais certamente treinar-te na arte de programar.

Assim, estudas as regras do jogo “Star Battle” (Secção 2) e, apesar de ainda não teres tido a cadeira de estruturas de dados, defines a estrutura de dados que utilizarás (Secção 3). Crias ainda um programa em Prolog (primeiras linhas na Secção 4) e resolves os predicados que te permitem resolver os puzzles (Secção 5). Sobre as condições de realização do projecto, a sua avaliação e recomendações, vê, sff, as Secções 6 e 7. Obrigada e diverte-te!

2 O puzzle

O puzzle consiste num tabuleiro (assuma-se uma matriz de dimensão NxN[1]), que tem inicialmente 0 ou mais estrelas em algumas posições. Esse tabuleiro está dividido em regiões. A Figura 1 mostra um exemplo de um puzzle inicial (neste exemplo, sem estrelas)[2].

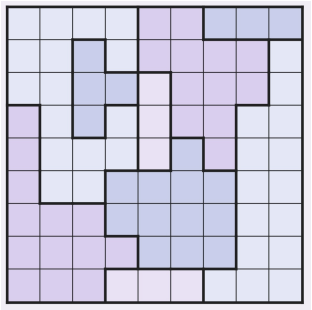

Figura 1: Exemplo de um puzzle inicial.

[1] Vamos trabalhar com matrizes de tamanho 9, mas o código deve ser genérico.
[2] Este desafio corresponde ao puzzle 9-2 do ficheiro puzzles.pl.

O objectivo é colocar duas estrelas em cada linha, em cada coluna e em cada região, tendo em conta que uma estrela não pode estar na **vizinhança** de outra outra estrela. Isto é, após ser colocada uma estrela, não pode existir outra nem na posição imediatamente acima, nem na posição imediatamente abaixo, nem na posição imediatamente à esquerda, nem na posição imediatamente à direita, nem nas posições diagonais. Sempre que se conclui que não pode existir uma estrela numa dada posição, deve inserir-se um ponto nessa posição.

A Figura 2 mostra o puzzle anterior quase resolvido.

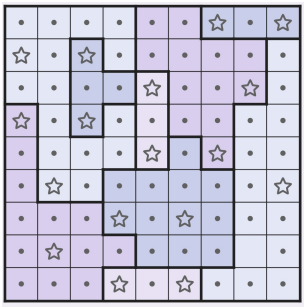

Figura 2: Exemplo de um puzzle quase resolvido (descobre o que falta).

3 Estruturas de dados

Decides que vais trabalhar com:

- Um tabuleiro (a preencher), constituído por uma matriz (uma lista de listas, NxN), em que cada lista representa uma linha do puzzle a resolver;
- Uma estrutura, constituído por uma matriz (também uma lista de listas), em que cada lista representa uma região do puzzle dado; esta estrutura deverá ser usada para consulta e não deve ser alterada.

Assim, as matrizes seguintes representam, respectivamente:

- o tabuleiro a preencher (“_” são variáveis em Prolog);
- a estrutura associada ao tabuleiro da Figura 1 (cada região é identificada por um número único). Por exemplo, a região identificada pelo número 9 representa as 3 casas a rosa a meio da última linha da Figura 1 (colunas 4, 5 e 6);
- o tabuleiro da Figura 2, em que “e” representa uma “estrela” e “p” um “ponto”.

```
[[_, _, _, _, _, _, _, _, _],
[_, _, _, _, _, _, _, _, _],
[_, _, _, _, _, _, _, _, _],
[_, _, _, _, _, _, _, _, _],
[_, _, _, _, _, _, _, _, _],
[_, _, _, _, _, _, _, _, _],
[_, _, _, _, _, _, _, _, _],
[_, _, _, _, _, _, _, _, _],
[_, _, _, _, _, _, _, _, _]]
```

```
[[1, 1, 1, 1, 2, 2, 3, 3, 3],
[1, 1, 4, 1, 2, 2, 2, 2, 5],
[1, 1, 4, 4, 6, 2, 2, 2, 5],
[7, 1, 4, 1, 6, 2, 2, 5, 5],
[7, 1, 1, 1, 6, 8, 2, 5, 5],
[7, 1, 1, 8, 8, 8, 8, 5, 5],
[7, 7, 7, 8, 8, 8, 8, 5, 5],
[7, 7, 7, 7, 8, 8, 8, 5, 5],
[7, 7, 7, 9, 9, 9, 5, 5, 5]]

[[p, p, p, p, p, p, e, p, e],
[e, p, e, p, p, p, p, p, p],
[p, p, p, p, e, p, p, e, p],
[e, p, e, p, p, p, p, p, p],
[p, p, p, p, e, p, e, p, p],
[p, e, p, p, p, p, p, p, e],
[p, p, p, e, p, e, p, p, p],
[p, e, p, p, p, p, p, _1064, p], % _1064 é uma variável
[p, p, p, e, p, e, p, p, p]]
```

4 O programa em Prolog

O teu ficheiro em Prolog (extensão pl) deverá ter as seguintes linhas iniciais:

```
% Escreve aqui o teu número e nome
:- use_module(library(clpfd)). % para poder usar transpose/2
:- set_prolog_flag(answer_write_options,[max_depth(0)]). % ver listas completas
:- [puzzles]. % Ficheiro dado. A avaliação terá mais puzzles.
:- [codigoAuxiliar]. % Ficheiro dado. Não alterar.
% Atenção: nao deves copiar nunca os puzzles para o teu ficheiro de código
% Segue-se o código
```

No ficheiro puzzles.pl que te é dado, tens o conjunto de puzzles que são usados neste enunciado. No ficheiro codigoAuxiliar.pl tens um conjunto de predicados que te poderão vir a ser úteis.

ATENÇÃO: não deves copiar NUNCA bocados de texto a partir deste pdf, correndo o risco de ficares com um input mal formatado (problema: muitas vezes não é visível).

5 Predicados a implementar

5.1 Visualização

Com o objectivo de conseguir visualizar melhor as matrizes em jogo, decides implementar os predicados `visualiza/1` e `visualizaLinha/1`, definidos como se segue.

– `visualiza(Lista)` é verdade se `Lista` é uma lista e a aplicação deste predicado permite escrever, por linha, cada elemento da lista `Lista`.
– `visualizaLinha(Lista)` é verdade se `Lista` é uma lista e a aplicação deste predicado permite escrever, por linha, cada elemento da lista `Lista`, aparecendo antes o número da linha em causa, um ":" e um espaço.

Por exemplo,

```
?- visualiza([1, a, _]).
1
a
_20280
true.

?- visualiza([[1, 7], [(a, 9), (4, 6)], _]).
[1,7]
[(a,9),(4,6)]
_23844
true.

?- visualizaLinha([1, a, _]).
1: 1
2: a
3: _25218
true.

?- visualizaLinha([[1, 7], [(a, 9), (4, 6)], _]).
1: [1,7]
2: [(a,9),(4,6)]
3: _28974
true.
```

Sugestão: carrega o teu código e faz:
?- sol(9-2, Tabuleiro), visualizaLinha(Tabuleiro).

5.2 Inserção de estrelas e pontos

De seguida, decides que queres controlar a inserção de objectos (estrelas (e) ou pontos (p)) no tabuleiro e decides implementar os predicados `insereObjecto/3`, `insereVariosObjectos/3`, `inserePontosVolta/2` e `inserePontos/2` definidos como se segue[3].

ATENÇÃO: um tabuleiro, enquanto não estiver terminado, é constituído por listas cheias de variáveis. Não precisarás de criar novos tabuleiros; apenas de instanciar (unificar), nesse tabuleiro, as variáveis com pontos ou estrelas. Se não sabes o que "unificar" quer dizer (shame on you!), vai já descobrir.

[3] Atenção à utilização do `nth1/3`, grande amigo de elegante polimodalidade, que falha quando a posição dada não faz parte da lista. Se não sabes o que "polimodalidade" quer dizer, vai rapidamente descobrir.

– `insereObjecto((L, C), Tabuleiro, Obj)` é verdade se `Tabuleiro` é um tabuleiro que após a aplicação deste predicado passa a ter o objecto `Obj` nas coordenadas `(L,C)`, caso nestas se encontre uma variável. De notar que o predicado NÃO deve falhar se as coordenadas não fizerem parte do tabuleiro ou se já existir um objecto, ponto (p) ou estrela (e), nessas coordenadas (nesses casos não faz nada).
– `insereVariosObjectos(ListaCoords, Tabuleiro, ListaObjs)` é verdade se `ListaCoords` for uma lista de coordenadas, `ListaObjs` uma lista de objectos e `Tabuleiro` um tabuleiro que, após a aplicação do predicado, passa a ter nas coordenadas de `ListaCoords` os objectos de `ListaObjs`. Estas duas listas têm o mesmo tamanho e devem ser percorridas ao mesmo tempo. Mais uma vez o predicado não deve falhar se alguma coordenada não fizer parte da matriz ou se já existir um objecto nessa coordenada. Neste caso deve avançar para a coordenada e objecto seguintes. No entanto, o predicado deve falhar se as listas `ListaCoords` e `ListaObjs` tiverem dimensões diferentes.
– `inserePontosVolta(Tabuleiro, (L, C))` é verdade se `Tabuleiro` é um tabuleiro que, após a aplicação do predicado, passa a ter pontos (p) à volta das coordenadas `(L, C)` (cima, baixo, esquerda, direita e diagonais).
– `inserePontos(Tabuleiro, ListaCoord)` é verdade se `Tabuleiro` é um tabuleiro que, após a aplicação do predicado, passa a ter pontos (p) em todas as coordenadas de `ListaCoord`. Se em alguma coordenada de `ListaCoord` existir um objecto que não seja uma variável, salta em frente.

Por exemplo[4],

```
?- Tab = [[_, e, _], [_, p, p], [_, _, p]], insereObjecto((1,3), Tab, ola).
Tab = [[_32746,e,ola],[_32770,p,p],[_32794,_32800,p]].

?- Tab = [[_, e, _], [_, p, p], [_, _, p]], insereObjecto((1,2), Tab, ola).
Tab = [[_37056,e,_37068],[_37080,p,p],[_37104,_37110,p]].

?- Tab = [[_, e, _], [_, p, p], [_, _, p]], insereObjecto((1,4), Tab, ola).
Tab = [[_41204,e,_41216],[_41228,p,p],[_41252,_41258,p]].

?- Tab = [[_, e, _], [_, p, p], [_, _, p]], insereObjecto((0,4), Tab, ola).
Tab = [[_45352,e,_45364],[_45376,p,p],[_45400,_45406,p]].

?- Tab = [[_, e, _], [_, p, p], [_, _, p]],
insereVariosObjectos([(1,3), (4,5), (1, 2), (1, 1)], Tab, [ola, ole, oi, batata]).
Tab = [[batata,e,ola],[_62460,p,p],[_62484,_62490,p]].

?- Tab = [[_, e, _], [_, p, p], [_, _, p]],
insereVariosObjectos([(1,3), (4,5), (1, 2)], Tab, [ola, ole]).
false.

Tab = [[_, e, _], [_, p, p], [_, _, p]], inserePontosVolta(Tab, (2, 1)).
Tab = [[p,e,_75260],[_75272,p,p],[p,p,p]].

?- Tab = [[_, e, _], [_, p, p], [_, _, p]], inserePontos(Tab, [(1,1), (2,1)]).
Tab = [[p,e,_82416],[p,p,p],[_82452,_82458,p]].
```

[4]Nota que nos exemplos seguintes, bem como em muitos neste enunciado, os tabuleiros apresentados são apenas ilustrativos e não puzzles a resolver. Em muitos dos casos nem seriam tabuleiros possíveis, pois alguns têm, por exemplo, estrelas sem pontos à volta, o que nunca deve acontecer.

5.3 Consultas

E pronto, já consegues ter um conjunto de funcionalidades que te permitem visualizar um tabuleiro e inserir estrelas ou pontos. Mas ainda não implementaste nada que te permita "consultar" o tabuleiro. Decides implementar os predicados `objectosEmCoordenadas/3`, `coordObjectos/5` e `coordenadasVars/2`. Este último poderá ser-te útil lá para a frente, para te ajudar a decidir quando deves parar o teu programa, porque o que implementaste não te permitirá evoluir mais (sim, será triste). Defines estes predicados como se segue.

– `objectosEmCoordenadas(ListaCoords, Tabuleiro, ListaObjs)` é verdade se `ListaObjs` for a lista de objectos (pontos, estrelas ou variáveis) das coordenadas `ListaCoords` no tabuleiro `Tabuleiro`, apresentados na mesma ordem das coordenadas. Neste caso, o predicado deve falhar se alguma coordenada não pertencer ao tabuleiro.
– `coordObjectos(Objecto, Tabuleiro, ListaCoords, ListaCoordObjs, NumObjectos)` é verdade se `Tabuleiro` for um tabuleiro, `Listacoords` uma lista de coordenadas e `ListaCoordObjs` a sublista de `ListaCoords` que contém as coordenadas dos objectos do tipo `Objecto`, tal como ocorrem no tabuleiro (o comportamento deverá ser o mesmo se o objecto pesquisado for uma variável). `NumObjectos` é o número de objectos `Objecto` encontrados. `ListaCoordObjs` deve estar ordenada por linha e coluna (sugestão: usar o `sort`).
– `coordenadasVars(Tabuleiro, ListaVars)` é verdade se `ListaVars` forem as coordenadas das variáveis do tabuleiro `Tabuleiro`. Mais uma vez, `ListaVars` deve estar ordenada por linhas e colunas.

Por exemplo,

```
?- Tab = [[_, e, _], [_, p, p], [_, _, p]],
objectosEmCoordenadas([(1, 1), (2, 3), (3, 3)], Tab, Obj).
...
Obj = [_41130,p,p].

?- Tab = [[_, e, _], [_, p, p], [_, _, p]],
objectosEmCoordenadas([(1, 1), (2, 4), (3, 3)], Tab, Obj).
false.

?- Tab = [[_, e, _], [_, p, p], [_, _, p]],
coordObjectos(e, Tab, [(1, 3), (3, 3)], LCO, Num).
...
LCO = [],
Num = 0.

?- Tab = [[_, e, _], [_, p, p], [_, _, p]],
coordObjectos(e, Tab, [(1, 2), (3, 3)], LCO, Num).
...
LCO = [(1,2)],
Num = 1.

?- Tab = [[_, e, _], [_, p, p], [_, _, p]],
 coordObjectos(_, Tab, [(1,1), (1, 2), (2, 1)], LCO, Num).
...
LCO = [(1,1),(2,1)],
Num = 2.

?- Tab = [[1, _], [3, _]], coordenadasVars(Tab, [(1, 2), (2, 2)]).
Tab = [[1,_14546],[3,_14564]].
```

```
?- Tab = [[1, _], [3, _]], coordenadasVars(Tab, [(1, 2), (1, 2)]).
false.
```

5.4 Estratégias

Chegou o momento mais esperado: implementar algumas estratégias para resolver os puzzles. Pensas[5] que se tudo estiver transformado em listas (linhas, colunas e estruturas – ver no codigoAuxiliar.pl o predicado `coordTodas/2`) é uma questão de:

- ir fechando as linhas, colunas e regiões, sempre que têm as duas estrelas, ou apenas espaço para a única estrela em falta ou apenas espaço para as duas únicas estrelas em falta; e
- ir encontrando padrões a preencher (por exemplo, três casas vazias sequenciais numa estrutura cheia de pontos leva a que se ponha uma estrela na primeira e na última posição; e, claro, os respectivos pontos à volta).

Com base neste raciocínio, implementas o que se segue.

5.4.1 Fechar linhas, colunas ou estruturas

Começas por:

- Implementar o `fechaListaCoordenadas/2`, que, dada uma lista de coordenadas (que representará as coordenadas de uma única linha, coluna ou região):
  - h1: sempre que a linha, coluna ou região associada à lista de coordenadas tiver 2 duas estrelas, enche as restantes coordenadas de pontos;
  - h2: sempre que a linha, coluna ou região associada à lista de coordenadas tiver uma única estrela e uma única posição livre, insere uma estrela na posição livre e insere pontos à volta da estrela;
  - h3: sempre que a linha, coluna ou região associada à lista de coordenadas não tiver nenhuma estrela e tiver duas únicas posições livres, insere uma estrela em cada posição livre e insere pontos à volta de cada estrela inserida;
- Implementar um predicado, o `fecha/2`, que recebe uma lista de listas (linhas, colunas ou regiões) e aplica o predicado anterior a cada lista.

Tendo em conta estas intuições, implementas o seguinte:

– `fechaListaCoordenadas(Tabuleiro, ListaCoord)` que é verdade se `Tabuleiro` for um tabuleiro e `ListaCoord` for uma lista de coordenadas; após a aplicação deste predicado, as coordenadas de `ListaCoord` deverão ser apenas estrelas e pontos, considerando as hipóteses definidas anteriormente (h1-h2-h3). Se nenhuma das hipóteses se verificar, o tabuleiro mantém-se inalterável (o predicado não falha).
– `fecha(Tabuleiro, ListaListasCoord)` que é verdade se `Tabuleiro` for um tabuleiro e `ListaListasCoord` for uma lista de listas de coordenadas. Após a aplicação deste predicado, `Tabuleiro` será o resultado de aplicar o predicado anterior a cada lista de coordenadas.

[5] É possível que não te ocorra nada disto ou que te ocorram coisas mais inteligentes. Mas as funcionalidades descritas são o que deves implementar.

Por exemplo (os seguintes exemplos são apenas ilustrativos; não são puzzles a resolver),

```
% Fecha linha
?- Tab = [[_, e, e], [_, p, p], [_, _, p]],
fechaListaCoordenadas(Tab, [(1, 1), (1, 2), (1, 3)]).
Tab = [[p,e,e],[_4176,p,p],[_4200,_4206,p]].

% Fecha coluna
?- Tab = [[_, e, e], [_, p, p], [_, _, p]],
fechaListaCoordenadas(Tab, [(1, 2), (2, 2), (3, 2)]).
Tab = [[_12046,e,e],[p,p,p],[p,e,p]].

% Fecha coluna e linha
?- Tab = [[_, e, e], [_, p, p], [_, _, p]],
fecha(Tab, [[(1, 2), (2, 2), (3, 2)], [(1, 1), (1, 2), (1, 3)]]).
Tab = [[p,e,e],[p,p,p],[p,e,p]].
```

A Figura 3 ilustra este exemplo:

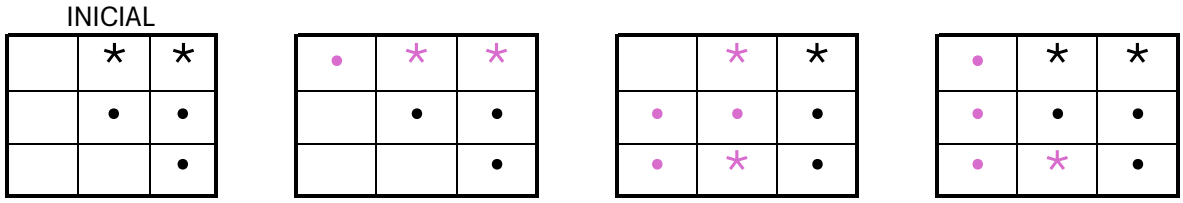

Figura 3: Puzzle inicial (não real) e os três casos do exemplo anterior.

5.4.2 Encontrar padrões

Entretanto, pensas que terás de encontrar padrões que te permitam inserir estrelas e pontos.

Um padrão possível é quando se encontra uma linha, coluna ou região (todos na sua forma de coordenadas), com N variáveis sequenciais (isto é, N coordenadas seguidas da lista estão por preencher). Na Figura 4 podes encontrar um conjunto de sequências identificadas nos puzzles que te permitirão de seguida aplicar os únicos padrões que serão abordados neste projecto: o padrão I e o padrão T (na figura do meio). Nota que o terceiro puzzle é um caso hipotético em que toda uma região estaria preenchida com pontos a não ser a sequência indicada.

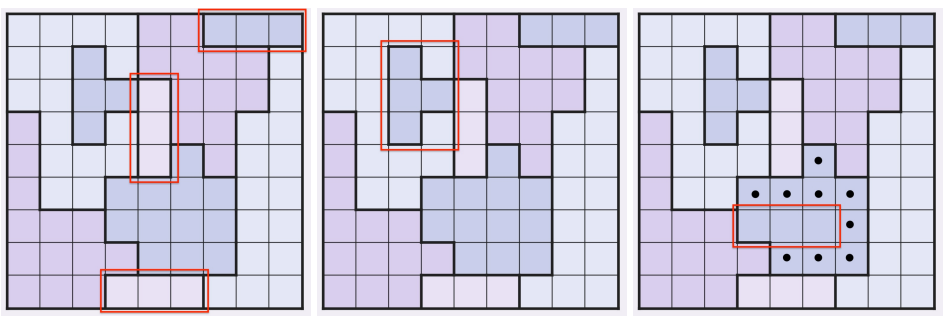

Figura 4: Procura de sequências às quais se poderão aplicar os padrões.

Decides então criar o predicado encontraSequencia/4.

– `encontraSequencia(Tabuleiro, N, ListaCoords, Seq)` que é verdade se `Tabuleiro` for um tabuleiro, `ListaCoords` for uma lista de coordenadas e `N` o tamanho de `Seq`, que é uma sublista de `ListaCoords` (portanto, com as coordenadas na mesma ordem) que verifica o seguinte:

- as suas coordenadas representam posições com variáveis;
- as suas coordenadas aparecem seguidas (numa linha, coluna ou região);
- `Seq` pode ser concatenada com duas listas, uma antes e uma depois, **eventualmente vazias ou com pontos nas coordenadas respectivas**, permitindo obter `ListaCoords`. De notar que se houver mais variáveis na sequência que N o predicado deve falhar.

Por exemplo (mais uma vez, os seguintes exemplos são apenas ilustrativos; não são puzzles a resolver),

```
?- Tab = [[_, e, e], [_, p, p], [_, _, p], [p, _, _]],
encontraSequencia(Tab, 3, [(1,1), (1,2), (1, 3)], Seq).
false.

?- Tab = [[_, e, e], [_, p, p], [_, _, p]],
encontraSequencia(Tab, 3, [(1,1), (2,1), (3, 1)], Seq).
...
Seq = [(1,1),(2,1),(3,1)].

?- Tab = [[_, e, e], [_, p, p], [_, _, p]],
encontraSequencia(Tab, 3, [(1,1), (3, 1), (2, 1)], Seq).
...
Seq = [(1,1),(3,1),(2,1)].

?- Tab = [[_, e, e], [_, p, p], [_, _, p], [p, _, _]],
encontraSequencia(Tab, 3, [(1,1), (2,1), (3, 1)], Seq).
...
Seq = [(1,1),(2,1),(3,1)].

?- Tab = [[_, e, e], [_, p, p], [_, _, p], [p, _, _]],
encontraSequencia(Tab, 2, [(1,1), (2,1), (3, 1)], Seq).
false

?- Tab = [[_, e, e], [_, p, p], [_, _, p]],
encontraSequencia(Tab, 3, [(1,1), (2,1), (3, 2)], Seq).
...
Seq = [(1,1),(2,1),(3,2)].
?- inicial(9, Tab), encontraSequencia(Tab, 3,  [(1,1), (1,2), (1, 3)], Seq).
...
Seq = [(1,1),(1,2),(1,3)].

?- inicial(9, Tab), encontraSequencia(Tab, 3,  [(1,1), (1,2), (1, 3), (1,4)], Seq).
false.

?- inicial(9, Tab), encontraSequencia(Tab, 4,  [(1,1), (1,2), (1, 3), (1,4)], Seq).
...
Seq = [(1,1),(1,2),(1,3),(1,4)].
```

Decides debruçar-te para o caso em que N = 3 (e chamas então I a esse padrão). Para isso implementas o predicado `aplicaPadraoI/2`:

– `aplicaPadraoI(Tabuleiro, [(L1, C1), (L2, C2), (L3, C3)])`, que é verdade se `Tabuleiro` for um tabuleiro e `[(L1, C1), (L2, C2), (L3, C3)]` for uma lista de coordenadas (por exemplo, uma sequência calculada anteriormente). Após a aplicação deste predicado, `Tabuleiro` será o resultado de colocar uma estrela (e) em `(L1, C1)` e `(L3, C3)` e os obrigatórios pontos (p) à volta de cada estrela.

Por exemplo (de novo, exemplos ilustrativos e não puzzles a resolver),

```
?- Tab = [[_, e, e], [_, p, p], [_, _, p]],
aplicaPadraoI(Tab, [(1,1), (2,1), (3, 1)]).
Tab = [[e,e,e],[p,p,p],[e,p,p]].

?- Tab = [[_, e, e], [_, p, p], [_, _, p], [_, _, _]],
aplicaPadraoI(Tab, [(4,1), (4,2), (4, 3)]).
Tab = [[_10568,e,e],[_10592,p,p],[p,p,p],[e,p,e]].

?- Tab = [[_, e, e], [_, p, p], [_, _, p]],
aplicaPadraoI(Tab, [(2,1), (3, 1)]).
false.

?- Tab = [[_, e, e], [_, p, p], [_, _, p]],
aplicaPadraoI(Tab, [(1,1), (2,1), (3, 1), (4,1)]).
false.

?- Tab = [[_, e, e, _], [_, p, p, _], [_, _, p, _], [_, _, _, _]],
aplicaPadraoI(Tab, [(1,1), (2,1), (3, 1), (4,1)]).
false.

?- Tab = [[_, e, e, _], [_, p, p, _], [_, _, p, _], [_, _, _, _]],
Tab = [[e,e,e,_12148],[p,p,p,_12178],[e,p,p,_12208],[p,p,_12232,_12238]].
```

Finalmente, pensas que tens de arranjar um predicado, o `aplicaPadroes/2`, genérico, que te permita descobrir padrões. Neste caso, aplicar o padrão I e também o padrão T que misteriosamente apareceu no teu codigoAuxiliar.pl. O padrão I deve ser aplicado a sequências de tamanho 3; o padrão T a sequências de tamanho 4. Decides implementas o seguinte:

– `aplicaPadroes(Tabuleiro, ListaListaCoords)` que é verdade se `Tabuleiro` for um tabuleiro, `ListaListaCoords` for uma lista de listas com coordenadas; após a aplicação deste predicado ter-se-ão encontrado sequências de tamanho 3 e aplicado o `aplicaPadraoI/2`, ou então ter-se-ão encontrado sequências de tamanho 4 e aplicado o `aplicaPadraoT/2`.

Observas de novo o puzzle anterior (ver Figura 5) e identificas as posições do padrão T – posições (2, 3), (3, 3), (3, 4) e (4, 3) – e dos 3 padrões I – posições (1, 7), (1, 8) e (1, 9), posições (3, 5), (4, 5) e (5, 5) e posições (9, 4), (9, 5) e (9, 6).

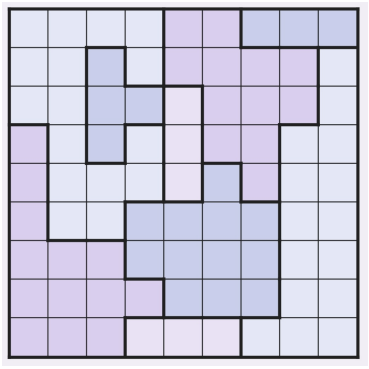

Figura 5: Puzzle com 1 padrão T e 3 padrões I.

Com base nesta figura, deve ser claro o tabuleiro que se obtém de seguida (variáveis simplificadas). Nota que para obteres todas as listas do tabuleiro que queres percorrer, isto é, para obteres todas as coordenadas em jogo (linhas, colunas e regiões), podes usar o predicado `coordTodas/2` do codigoAuxiliar.pl.

```
?- regioes(9-2, E), inicial(9, Tab), coordTodas(E, CT), aplicaPadroes(Tab, CT).
...
Tab = [[_,p,p,p,_,p,e,p,e],
[_,p,e,p,p,p,p,p,p],
[_,p,p,p,e,p,_,_,_],
[_,p,e,p,p,p,_,_,_],
[_,p,p,p,e,p,_,_,_],
[_,_,_,p,p,p,_,_,_],
[_,_,_,_,_,_,_,_,_],
[_,_,p,p,p,p,p,_,_],
[_,_,p,e,p,e,p,_,_]],
```

Este resultado corresponde ao puzzle da Figura 6:

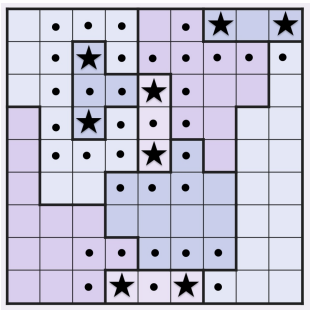

Figura 6: Puzzle com 1 padrão T e 3 padrões I, após a aplicação do `aplicaPadroes`.

5.5 Apoteose Final

É o momento de juntar as peças que tens. Já te fartaste de implementar predicados, mas compreendes que o que implementaste não te permitirá resolver todos as “Star Battles". No entanto, é hora de fechar a loja e atacas o predicado `resolve/2`.

– `resolve(Estruturas, Tabuleiro)` é verdade se `Estrutura` for uma estrutura e `Tabuleiro` for um tabuleiro que resulta de aplicar os predicados `aplicaPadroes/2` e `fecha/2` até já não haver mais alterações nas variáveis do tabuleiro; em alguns casos, este predicado resolve o desafio até ao fim (não, não vai ser preciso usar a tentativa e erro do Prolog); noutros não.

Por exemplo[6],

```
?- regioes(9-2, E), inicial(9, T), resolve(E, T).
...
T = [[p,p,p,p,p,p,e,p,e],
[e,p,e,p,p,p,p,p,p],
[p,p,p,p,e,p,_746,_752,_758],
[e,p,e,p,p,p,p,p,p],
[p,p,p,p,e,p,_866,_872,_878],
[p,e,p,p,p,p,p,_932,_938],
[p,p,p,e,p,e,p,p,p],
[p,e,p,p,p,p,p,_1052,_1058],
[p,p,p,e,p,e,p,p,p]]. %oh, ainda tem vars

?- regioes(9-2, E), inicial(9-2-1, T), resolve(E, T).
...
T = [
[p,p,p,p,p,p,e,p,e],
[e,p,e,p,p,p,p,p,p],
[p,p,p,p,e,p,p,e,p],
[e,p,e,p,p,p,p,p,p],
[p,p,p,p,e,p,e,p,p],
[p,e,p,p,p,p,p,p,e],
[p,p,p,e,p,e,p,p,p],
[p,e,p,p,p,p,p,e,p],
[p,p,p,e,p,e,p,p,p]]. % que lindinho

?- regioes(9-24, E), inicial(9-24-1, T), resolve(E, T).
...
T = [
[p,p,p,p,e,p,p,e,p],
[e,p,e,p,p,p,p,p,p],
[p,p,p,p,p,e,p,e,p],
[p,e,p,e,p,p,p,p,p],
[p,p,p,p,p,p,e,p,e],
[p,p,e,p,e,p,p,p,p],
[e,p,p,p,p,p,e,p,p],
[p,p,p,e,p,p,p,p,e],
[p,e,p,p,p,e,p,p,p]]. % fofinho
```

Nota: se quiseres implementar mais estratégias, estás à vontade. No entanto, os resultados que alcançares com novas estratégias não podem colidir com os testes de avaliação automática.

Entretanto, acabaste o projecto e vem-te à cabeça (ou talvez não; vamos imaginar) o seguinte poema de Ricardo Reis[7]:

[6] Os resultados são apresentados linha a linha por uma questão de legibilidade.
[7] Se não sabes quem é, vai descobrir; também é importante

```
Para ser grande, sê inteiro:
Nada teu exagera ou exclui.

Sê todo em cada coisa.
Põe quanto és no mínimo que fazes.

Assim em cada lago a lua toda brilha, porque alta vive.
```

Não é sobre estrelas, mas refere a lua. E não resistes: vais à procura de uma *app* sobre a lua.

6 Avaliação e entrega

6.1 Cotação

A nota do projecto será baseada no seguinte:

- Execução correcta – 16 valores distribuídos da seguinte forma:
  1. Visualização (1.0 valores)
     (a) `visualiza/1`: 0.5 valores;
     (b) `visualizaLinha/1`: 0.5 valores;
  2. Inserção (4.0)
     (a) `insereObjecto/3`: 1.0 valor;
     (b) `insereVariosObjectos/3`: 1.0 valor;
     (c) `inserePontosVolta/2`: 1.0 valor;
     (d) `inserePontos`: 1.0 valor;
  3. Consultas (3.5 valores)
     (a) `objectosEmCoordenadas/3`: 1 valor
     (b) `coordObjectos/5`: 1.5 valor;
     (c) `coordenadasVars/2`: 1 valor
  4. Estratégias – parte 1 (6.0 valores)
     (a) `fechaListaCoordenadas/2`: 1.5 valores;
     (b) `fecha/2`: 0.5 valores;
     (c) `encontraSequencia/4`: 2.0 valores;
     (d) `aplicaPadraoI/2`: 1.5 valores;
     (e) `aplicaPadroes/2`: 0.5 valores;
  5. Apoteose final (1.5 valores)
     (a) `resolve`: 1.5 valor.
- Estilo de programação e facilidade de leitura – 4 valores assim distribuídos:
  - Comentários (1.5 valores): deverão ser incluídos comentários para o utilizador (descrição sumária do predicado); deverão também ser incluídos, quando se justifique, comentários para o programador;
  - Código legível (0.5 valores): não devemos ter de fazer *scroll* para ler uma linha;

  - Implementação de predicados não excessivamente longos (1.0 valor): uma má abstração procedimental e duplicações de código serão penalizados. No entanto, o facto de usar um predicado recursivo, desde que bem feito, não é penalizado em relação ao uso de predicados funcionais;
  - Boa escolha de nomes dos predicados auxiliares e das variáveis (1.0 valor): os nomes dos predicados e variáveis deverão tornar o programa mais compreensível.

Presença de *warnings* serão penalizadas (-2 valores).

6.2 Condições de realização e prazos

O projecto é realizado individualmente. O código do projecto deve ser entregue obrigatoriamente até às **23h59 do dia 13 de Janeiro de 2025**. Depois desta hora, não serão aceites projectos sob pretexto algum. A ter em conta:

- Cada estudante terá acesso a um repositório git e deverá submeter o seu código no ficheiro "projecto.pl" disponível no seu repositório. O ficheiro de código deve conter, em comentário na primeira linha, o número e o nome do aluno, tal como indicado anteriormente;
- A avaliação da execução do código do projecto será feita automaticamente no GitLab, usando vários testes configurados no sistema. O tempo de execução de cada teste está limitado, bem como a memória utilizada. Tipicamente, só se poderá efectuar uma nova submissão 15 minutos depois da submissão anterior. De notar que, se for feita uma submissão no GitLab a menos de 15 minutos do prazo de entrega, qualquer outra submissão posterior do estudante não será avaliada;
- **O limite de submissões sem desconto é de 30; a partir da submissão 31 serão descontados 0.1 valores (em 20) por cada submissão**;
- Não esquecer de remover as mensagens escritas no ecrã.

Serão publicadas em breve, na página da cadeira, as instruções detalhadas necessárias para a submissão do código no GitLab e a partir dessa altura será possível a submissão por via electrónica, sendo utilizada para efeitos de avaliação a última entrega efectuada.

Pode ou não haver uma discussão oral do projecto e/ou uma demonstração do funcionamento do programa (será decidido caso a caso).

6.3 Sobre as cópias

Os alunos deverão fazer o seu próprio código. Projectos muito semelhantes levarão à reprovação na disciplina e levantamento de processo disciplinar. Os alunos deverão **obrigatoriamente preencher um formulário** (detalhes em breve) a dizer que não usaram o ChatGPT ou outro *Large Language Model* (LLM) qualquer ou então, caso tenham usado (o que não aconselhamos), a explicar como decorreram as interações relativas a CADA predicado. Caso dois alunos tenham usado um LLM, não o indicarem, e os projectos forem considerados muito semelhantes, serão considerados cópia. O corpo docente da disciplina será o único juiz do que se considera copiar.

7 Recomendações

- Usem o SWI PROLOG, que vai ser usado para a avaliação do projecto.
- Durante o desenvolvimento do programa não se esqueçam da Lei de Murphy:
  - Todos os problemas são mais difíceis do que parecem;
  - Tudo demora mais tempo do que pensamos;
  - Se alguma coisa puder correr mal, vai correr mal, na pior das alturas possíveis.