Ficha de laboratório Nº 7: Apoio ao 1º Projeto

Inteligência Artificial - Escola Superior de Tecnologia de Setúbal

Prof. Joaquim Filipe
Eng. Filipe Mariano

Nota prévia

Os exercícios a desenvolver no âmbito da presente série de exercícios requerem uma leitura prévia do enunciado do 1º projeto. Se porventura ainda não o leram, devem fazê-lo antes de realizar o laboratório.

Objetivos da ficha

Este laboratório tem como objetivo desenvolver algumas das funções necessárias para o 1º projeto de procura em espaço de estados. Ao longo do mesmo, irá implementar:

- Seletores
- Funções Auxiliares
- Operadores

Os exercícios propostos neste laboratório devem servir de base para a resolução da 1ª fase do projeto. Contudo, não é necessário que a abordagem que cada grupo siga seja exatamente a mesma da que é proposta neste laboratório.

1. Representação do Problema

Gerar um tabuleiro do Jogo do Cavalo

1. Descarregar o ficheiro **laboratorio7.lisp** disponível no Moodle.
2. Abrir o ficheiro no IDE *LispWorks*.
3. Observar a estrutura de comentários presente no inicio do ficheiro para indicar o seu conteúdo e autor.
4. Neste ficheiro estão definidas as funções `tabuleiro-teste` e `tabuleiro-jogado` que retornam tabuleiros a testar o **Jogo do Cavalo**.
5. O tabuleiros retornados servirão de exemplo para os exercícios propostos neste laboratório. São tabuleiros de acordo com as regras do enunciado do projeto, de dimensão 10 x 10.
6. Compilar o código do ficheiro e executar as funções no *Listener*.

2. Exercícios

Seletores

1. `linha`: Função que recebe um índice e o tabuleiro e retorna uma lista que representa essa linha do tabuleiro.

```
CL-USER > (linha 0 (tabuleiro-teste))
(94 25 54 89 21 8 36 14 41 96)
```

2. `celula`: Função que recebe dois índices e o tabuleiro e retorna o valor presente nessa célula do tabuleiro.

```
CL-USER > (celula 0 1 (tabuleiro-teste))
25
```

Tabuleiro Aleatório

3. `lista-numeros`: Função que recebe um número positivo `n` e cria uma lista com todos os números entre 0 (inclusivé) e o número passado como argumento (exclusivé). Por *default* o `n` é 100.

```
CL-USER > (lista-numeros)
(99 98 97 96 95 94 93 92 91 90 89 88 87 86 85 ... 5 4 3 2 1 0)
```

4. `baralhar`: Função que recebe uma lista e irá mudar aleatoriamente os seus números. Faça uma função recursiva e utilize a função `nth`, a função `random` e a função `remover-se` do laboratório nº5 em que a função deve remover o número da lista igual ao encontrado aleatoriamente.
    - A condição de paragem é a lista estar vazia;
    - Deve utilizar a instrução `let` para guardar localmente o número encontrado numa posição aleatória através da seguinte instrução: `(nth (random (length lista)) lista)`
    - Através da função `remover-se` deverá remover da lista que está a ser passada como argumento na função recursiva, o número que foi encontrado aleatoriamente e que está guardado localmente.

```
CL-USER > (baralhar (lista-numeros))
(23 16 22 65 8 18 59 94 6 66 53 96 25 97 74 34 31 17 5 36 89 54 87 62 69 72 9 29
35 44 26 86 78 92 39 99 52 11 4 38 ...)
;este é um mero exemplo de output e a probabilidade de obter um output idêntico é
muito reduzida. Serve apenas para exemplificar que a lista anterior com os 100
números se encontra "baralhada".
```

5. `tabuleiro-aleatorio`: Copie a seguinte função para o *Editor* e experimente executá-la com uma lista de 100 números "baralhada" e em que o tamanho-linha é 10. O objetivo desta função é pegar numa lista e criar sublistas de `n` elementos recorrendo à função `subseq` que tem um comportamento semelhante ao substring para strings.

    - Substituir os parâmetros **lista** e **n** pela instrução que permite ter valores por *default*. Assuma que por *default* a **lista** será o resultado obtido na alínea 4 `(baralhar (lista-numeros))` e o **n** é `10`.

```
(defun tabuleiro-aleatorio (lista n)
  (cond
   ((null lista) nil)
```

```
    (t (cons (subseq lista 0 n) (tabuleiro-aleatorio (subseq lista n) n)))
    )
)
```

```
CL-USER > (tabuleiro-aleatorio)
((68 76 77 27 38 48 96 42 55 84) (86 43 73 97 72 66 35 52 51 71) (39 22 49 57 61
99 64 36 11 19) (31 40 87 41 90 7 81 65 75 85) (46 56 8 98 12 10 18 6 4 54) (88 91
45 15 92 53 25 14 94 58) (28 47 2 59 79 21 70 3 20 69) (44 5 80 33 82 60 30 74 37
83) (62 95 29 16 93 26 24 23 78 67) (9 13 32 17 50 1 0 89 63 34))
;este é um mero exemplo de output e a probabilidade de obter um output idêntico é
muito reduzida. Serve apenas para exemplificar que já temos um tabuleiro gerado
aleatoriamente.
```

Funções auxiliares

6. `substituir-posicao`: Função que recebe um índice, uma lista e um valor (por *default* o valor é NIL) e substitui pelo valor pretendido nessa posição.

```
CL-USER > (substituir-posicao 0 (linha 0 (tabuleiro-teste)))
(NIL 25 54 89 21 8 36 14 41 96)

CL-USER > (substituir-posicao 0 (linha 0 (tabuleiro-teste)) T)
(T 25 54 89 21 8 36 14 41 96)
```

7. `substituir`: Função que recebe dois índices, o tabuleiro e um valor (por *default* o valor é NIL). A função deverá retornar o tabuleiro com a célula substituída pelo valor pretendido. Utilize a função `substituir-posicao` definida anteriormente.

```
CL-USER > (substituir 0 0 (tabuleiro-teste) T)
(
    (T 25 54 89 21 8 36 14 41 96)
    (78 47 56 23 5 49 13 12 26 60)
    (0 27 17 83 34 93 74 52 45 80)
    (69 9 77 95 55 39 91 73 57 30)
    (24 15 22 86 1 11 68 79 76 72)
    (81 48 32 2 64 16 50 37 29 71)
    (99 51 6 18 53 28 7 63 10 88)
    (59 42 46 85 90 75 87 43 20 31)
    (3 61 58 44 65 82 19 4 35 62)
    (33 70 84 40 66 38 92 67 98 97)
)
```

8. `posicao-cavalo`: Função que recebe o tabuleiro e devolve a posição `(i j)` em que se encontra o cavalo. Caso o cavalo não se encontre no tabuleiro deverá ser retornado NIL.

```
CL-USER > (posicao-cavalo (tabuleiro-teste))
NIL

CL-USER > (posicao-cavalo (tabuleiro-jogado))
(0 0)
```

Operadores

Os operadores representam os movimentos possíveis num determinado estado. Para o Problema do Cavalo o máximo de movimentos possíveis serão 8, desde que essas casas não tenham sido ainda visitadas ou removidas pela regra dos simétricos ou duplos.

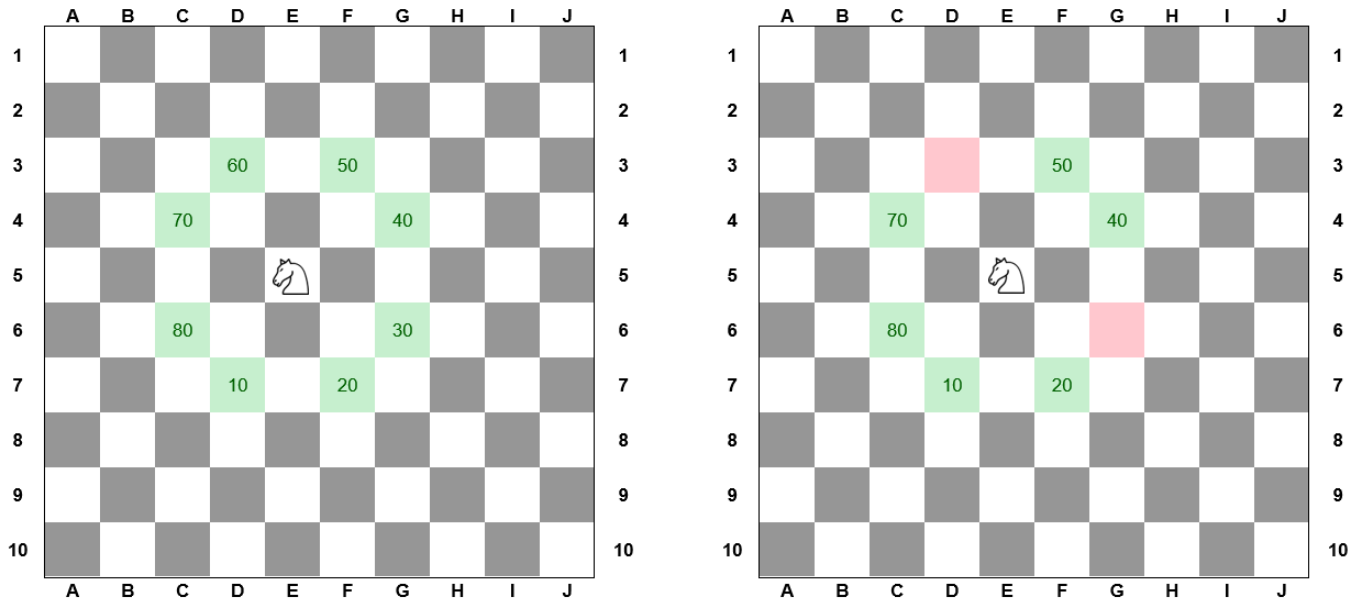

Figura 1: No 1º tabuleiro encontra-se uma situação em que os 8 movimentos possíveis poderão ser realizados porque as casas ainda não foram visitadas (têm pontos). No 2º tabuleiro apenas é possível realizar 6 movimentos porque duas das casas já foram visitadas.

Tal como ocorre para o problema das vasilhas do laboratório anterior, o número de movimentos para solucionar este puzzle é fixo (**8 movimentos**), e como tal pode optar por utilizar uma lista de operadores com `operador-1`, `operador-2`, `operador-3`, `operador-4`, `operador-5`, `operador-6`, `operador-7` e `operador-8`.

Por uma questão de uniformização da numeração do operador e qual o movimento que representa, vamos considerar que de acordo com a Figura 1 o `operador-1` faz o movimento para a casa de valor 10, o `operador-2` para a casa de valor 20, etc.

9. `operador-1`: Função que recebe o tabuleiro e movimenta o cavalo para a posição 2 linhas abaixo e uma coluna ao lado direito, ou seja, o movimento que de acordo com a Figura 1 leva o cavalo para a casa de valor 10.

```
CL-USER > (operador-1 (tabuleiro-jogado))
(
    (NIL 25 54 89 21 8 36 14 41 96)
    (78 47 56 23 5 NIL 13 12 26 60)
    (0 T 17 83 34 93 74 52 45 80)
```

```
    (69 9 77 95 55 39 91 73 57 30)
    (24 15 22 86 1 11 68 79 76 72)
    (81 48 32 2 64 16 50 37 29 71)
    (99 51 6 18 53 28 7 63 10 88)
    (59 42 46 85 90 75 87 43 20 31)
    (3 61 58 44 65 82 19 4 35 62)
    (33 70 84 40 66 38 92 67 98 97)
)
```

Nota: De salientar que o operador não está terminado e devem contemplar as seguintes situações para o vosso projeto:

1. Verificar se o movimento é válido;
2. Verificar se o cavalo está nalguma casa do tabuleiro, caso contrário deverá posicionar o cavalo numa casa da 1ª linha;
3. Após o movimento do cavalo, deverá aplicar a regra do número simétrico ou a regra do número duplo.