3 2a Fase

Nesta segunda fase do projecto vamos implementar algumas estratégias de procura não informada.

3.1 Calcular a lista de todos os estados seguintes possíveis (2val)

Criar uma função nextStates que dado um estado devolve uma lista de todos os estados seguintes possíveis.

```
"Exercise 2.1 - nextStates "
> (nextStates <state>)
(<next state 1> <next state 2> ... <next state n>)
```

3.2 Procura em profundidade limitada (Limited Depth-First Search) (2val)

Criar uma função limdepthfirstsearch que dado um problema e um limite faz uma procura em profundidade limitada. Termina devolvendo a lista de estados desde o estado inicial até ao estado objectivo encontrado, no caso de não conseguir atingir um estado objectivo devolve NIL se não tiverem sido cortados estados durante a procura e corte se durante a procura tenha havido pelo menos um estado sujeito a corte.

Exemplo:

```
"Exercise 2.2 - limdepthfirstsearch"
> (limdepthfirstsearch <problem> <limit>)
(<initial state> ... <goal state>)
```

3.3 Procura em profundidade iterativa (Iterative Depth-First Search) (2val)

Criar a função iterlimdepthfirstsearch que dado um problema faz uma procura em profundidade iterativa. Termina devolvendo a lista de estados desde o estado inicial até ao estado objectivo encontrado ou NIL no caso de não conseguir atingir um estado objectivo.

Exemplo:

```
"Exercise 2.3 - iterlimdepthfirstsearch"
> (iterlimdepthfirstsearch <problem>)
(<initial state> ... <goal state>)
```

Note que, contrariamente à primeira fase, haverá mais testes para além destes, usando outras situações e outros problemas VectorRacer.

4

Nesta segunda fase do projecto vamos implementar estratégias de procura informada e respectivas heurísticas.

4.1 Cálculo de heuristica (3val)

Criar um a função compute-heuristic que dado um estado estima a sua distância ao estado objectivo mais próximo. Para este caso específico vamos usar para o cálculo da heurística a solução de um problema com uma dinâmica simplificada. A dinâmica simplificada vai ignorar a inercia do carro, ou seja:

$$\begin{array}{rcl} p^{t+1} & = & p^t + a \\ p^{t+1}_l & = & p^t_l + a_l \\ p^{t+1}_c & = & p^t_c + a_c \end{array}$$

O resultado da função deverá ser lista de lista equivalente ao elemento env da estructura track onde para cada lo calização mostrar-se-á a distância dessa localização até ao estado final.

```
"Exercise 3.1 - compute-heuristic"
> (compute-heuristic <state>)
<integer>
```

A seguir mostra-se o cálculo da heuristica para cada uma das posições de uma pista sabendo que as posições que representam a meta têm o valor 0, as posições que representam obstáculos têm o valor <M> que correspondem ao most-positive-fix-num, sendo que todas as outras posições têm um inteiro que representa a sua distância à posição da meta mais próxima.

```
(<M> <M> <M> <M> <M> 10
           9
              7
                  5
                     3 2 <M> <M> <M> <M> <M>)
             8
                6
                   4
             8
               7
(<M> <M> <M> <M> <M>
         10
           9
                6
                  5
                    4
                     3
                       2
                        1 <M> <M> <M>)
(<M> 14 13 12 11 10
           9 <M> <M> <M> <M> <M> <M>
                        1 0 0 (M>)
                     3
                       2
0 <M>)
0
                            0 <M>)
                        1
```

4.2 Procura guiada (A*) (4val)

Criar a função a* que dado um problema faz uma procura A* usando a função heurística do problema. Termina devolvendo a lista de estados desde o estado inicial até ao estado objectivo encontrado ou NIL no caso de não conseguir atingir um estado objectivo.

```
"Exercise 3.2 - a*"
> (a* <problem>)
(<initial state> ... <goal state>)
```

4.3 Procura guiada e optimizada (3val)

Criar a função best-search que faz uma procura guiada com todas as optimizações que considere adequadas para melhorar o desempenho do seu programa. Termina devolvendo a lista de estados desde o estado inicial até ao estado objectivo encontrado ou NIL no caso de não conseguir atingir um estado objectivo.

```
"Exercise 3.3 - best-search"
> (best-search <problem>)
(<initial state> ... <goal state>)
```