

Projecto Inteligência Artificial (LEIC 3º Ano, 1º Semestre 2016/2017)

Tagus: <https://fenix.tecnico.ulisboa.pt/disciplinas/IArt9179/2016-2017/1-semestre>

Alameda:<https://fenix.tecnico.ulisboa.pt/disciplinas/IArt45179/2016-2017/1-semestre>

Manuel Lopes (Tagus) manuel.lopes@tecnico.ulisboa.pt
Ernesto Morgado (Alameda) emorgado@siscog.pt

October 20, 2016

Abstract

Neste projecto vamos desenvolver vários algoritmos para resolver um jogo. O projecto terá 3 fases incluindo: i) criação de ferramentas básicas; ii) desenvolvimento de métodos de procura não guiada para problemas pequenos; iii) desenvolvimento de métodos de procura guiada e optimização do método de resolução de forma a conseguir resolver problemas de maior dimensão. O jogo é o VectorRacer e a entrega e correção vai ser feita de forma automática usando o sistema Mooshak.

1 Descrição do jogo : VectorRacer

Este jogo VectorRacer (também chamado RaceTrack ou formula 1 de papel) é uma corrida de carros com uma dinâmica muito simplificada e que pode ser feito com caneta e papel quadriculado. Para uma descrição e história ver [https://en.wikipedia.org/wiki/Racetrack_\(game\)](https://en.wikipedia.org/wiki/Racetrack_(game))

1.1 Movimento e Regras

O movimento do carro é muito simples e respeita uma dinâmica de primeira ordem em que as acções são equivalentes a um conjunto de acelerações. Em cada instante é possível acelerar -1, 0 ou 1 unidades em cada direção. Ou seja o conjunto de acelerações em cada direção é $A = \{-1, 0, +1\}$ e portanto $a = (a_l, a_c)$ com $a_l, a_c \in A$.

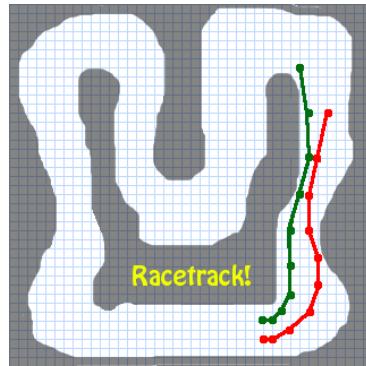


Figure 1: Exemplo de corrida com as primeiras jogadas de 2 jogadores.

Se o carro na jogada t esta na posição $p^t = (p_l, p_c)$ com a velocidade $v^t = (v_l, v_c)$, na jogada a seguir ele vai estar na posição:

$$p^{t+1} = p^t + v^t + a$$

Detalhadamente para cada direcção:

$$\begin{aligned} p_l^{t+1} &= p_l^t + v_l^t + a_l \\ p_c^{t+1} &= p_c^t + v_c^t + a_c \end{aligned}$$

Considera-se a notação (linha, coluna) para os vectores.

Considera-se a notação (linha, coluna) para os vectores.

Sendo $a = (a_l, a_c)$ a aceleração, a nova velocidade será então:

$$\begin{aligned} v_l^{t+1} &= v_l^t + a_l \\ v_c^{t+1} &= v_c^t + a_c \end{aligned}$$

Se o carro sair da pista ele volta a posição anterior com velocidade zero.

$$\begin{aligned} \text{if not ontrack}(p) \\ p^{t+1} &= p^t \\ v^{t+1} &= (0, 0) \end{aligned}$$

Quando o carro sai da pista há uma penalização de -20 pontos. A cada acção corresponde uma penalização de -1 ponto. Chegar a um dos estados da meta dá uma bonificação de 100 pontos.

A corrida termina quando o carro toca numa das posição marcada com meta, independentemente da velocidade.

O objectivo é chegar à meta obtendo a maior pontuação possível. Como as estratégias de procura estão orientadas para obter a solução de menor custo vamos considerar os valores simétricos dos apresentados acima para os custos das acções:

$$\begin{aligned} \text{movimento 1} \\ \text{saída de pista } 20 \\ \text{chegar à meta } -100 \end{aligned}$$

1.2 Bibliografia e ambiente de desenvolvimento

A matéria teórica necessária ao desenvolvimento do projecto pode ser encontrada no livro de texto adoptado [3]. O projecto deve ser implementado em Common Lisp garantindo a sua compatibilidade com o CLISP versão 2.49 [2]. A documentação do Common Lisp pode ser consultada em <http://www.lispworks.com/documentation/HyperSpec/Front/> [1]

1.3 Estruturas de dados

O ficheiro `datastructures.lisp` inclui a estrutura que define uma pista, a definição do estado do carro e a definição do problema que deverão ser usadas no projecto.

1.3.1 Definição de uma pista

```
;;; Definition of track
;;; * size - size of track
;;; * env - track where nil are obstacles, everything else is the track
;;; * startpos - initial position
;;; * endpositions - valid final positions
(defstruct track
  size
  env
  startpos
  endpositions)
```

O tamanho da pista é representado em Common Lisp por uma lista com dois elementos em que o primeiro elemento representa o número de linhas e o segundo o número de colunas.

A pista é representada por uma lista de listas. Cada elemento da lista representa uma linha completa em que cada elemento dessa lista representa o que está numa dada coluna. Os obstáculos são representados por NIL e as posições que podem ser usadas pelos carros por T.

A posição inicial é representada como uma lista de dois elementos em que o primeiro representa a linha e o segundo a coluna.

A linha da meta é representada por uma lista de posições.

O ficheiro `track0.lisp` inclui um exemplo do código lisp necessário para criar uma pista na representação descrita acima.

```
(setf *env* '((nil nil nil)
              (nil nil nil nil t t t t t t nil nil nil nil)
              (nil nil nil nil t t t t t t t t nil nil nil nil)
              (nil t t t t nil nil nil nil nil t t t t t nil)
              (nil t t t t nil nil nil nil nil nil t t t t nil)
              (nil t t t t nil nil nil nil nil nil nil t t t nil)
              (nil nil nil)))
```

```
(setf *track0*
      (make-track :size (list (length *env*)
                               (length (first *env*)))
                  :env *env*
                  :endpositions (orderlistofcoordinates '((3 15) (4 15) (5 15) (3 16) (4 16) (5 16)))
                  :startpos '(4 1)))
```

A função `orderlistofcoordinates` poder-se-a revelar muito útil para se poderem comparar estruturas do tipo `track`.

Uma função que seja capaz de ler ficheiros de pista `*.txt` e que transforme em `structtrack` será muito útil para poder testar pistas diferentes.

Deverá existir uma função que leia ficheiros de texto com a representação externa de uma pista. É fornecido um exemplo de uma pista no ficheiro `track0.txt`

```
XXXXXXXXXXXXXXXXXX
XXXXX000000000XXX
XXXXX00000000000XXX
X000000XXXXX000EEX
XS0000XXXXXX00EEX
X0000XXXXXXXXX0EEX
XXXXXXXXXXXXXXXXXX
```

Aqui 0 representa a pista, X são os obstáculos, S é a posição inicial e E são as posições finais.

1.3.2 Estado do carro

```
;;; State of the car
;;; * pos - position
;;; * vel - velocity
;;; * action - action that was used to generate the state
;;; * cost - cost of the action
;;; * track - VectorRace track
;;; * other - additional information
(defstruct state
  pos
  vel
  action
  cost
  track
  other)
```

A posição em que o carro se encontra é representada em Common Lisp como uma lista de dois elementos em que o primeiro representa a linha e o segundo a coluna, por exemplo (0 3) representa a posição na linha 0 coluna 3. A origem das coordenadas é a posição (0 0) e representa o canto superior esquerdo de uma pista.

A velocidade a que o carro se movimenta é representada em Common Lisp como uma lista de dois elementos em que o primeiro representa a velocidade na direcção horizontal e o segundo a velocidade na direcção vertical. Velocidades positivas deslocam o carro em direcções nas quais o valor das coordenadas é crescente, enquanto que velocidades negativas deslocam o carro em direcções nas quais o valor das coordenadas é negativo.

A acção que deu origem ao estado é representada por uma lista de dois elementos em que o primeiro representa a aceleração na direcção horizontal e o segundo a aceleração na direcção vertical, tal como para a velocidade.

O custo representa o custo da acção que originou a transição de estado. Atenção aqui é o custo da acção e não o custo acumulado.

A pista é representada tal como descrito no ponto anterior.

O campo `other` pode servir para guardar outra informação que considere relevante.

1.3.3 Estrutura de um problema

```
;;; Definition of a problem
;;; * initial-state
;;; * fn-nextstate - function that computes the successors of a state
;;; * fn-isGoal - function that identifies a goal state
;;; * fn-h - heuristic function
(defstruct problem
  initial-state
  fn-nextStates
  fn-isGoal
  fn-h)
```

Um problema tem que ter definido qual o estado inicial.

Uma função que gera os sucessores de um estado, ou seja uma função que aplicada a um estado gera uma lista com todos os estados que podem ser atingidos a partir desse.

Uma função que permite identificar um estado objectivo em que o jogo terminou.

Para as procuras guiadas é necessário também uma função heurística que aplicada a um estado estima a distância desse estado ao estado objectivo mais próximo.

2 1a Fase (2 valores)

Nesta primeira fase iremos construir várias ferramentas que servirão para nos ambientar-mos com as regras do jogo, à linguagem Common LISP e ao sistema de submissão e correção Mooshak¹.

2.1 Verificar se há um obstáculo numa dada posição da pista (0,4val)

Criar uma função `isObstaclep` que dada uma posição e uma pista devolve o valor lógico T se existir um obstáculo na posição e NIL no caso contrário.

```
"Exercise 1.1 - isObstaclep "
> (isObstaclep <pos> <track>)
NIL
```

2.2 Verificar se um estado é objectivo (0,4val)

Criar uma função `isGoalp` que dado um estado devolve o valor lógico T se o estado for objectivo e NIL caso não seja.

```
"Exercise 1.2 - isGoalp"
> (isGoalp <state>)
T
```

2.3 Calcular o estado seguinte dada uma acção (1,2val)

Criar uma função `nextState` que dado um estado e uma acção devolve o estado que resulta de aplicar a acção ao estado fornecido.

```
"Exercise 1.3 - nextState"
> (nextState <state> <action>)
<next state>
```

2.4 Funções de teste

Deverão ser implementadas as funções, e tantas outras quantas necessárias, no ficheiro `SolF1.lisp`. Ao executar o ficheiro `projF1.lisp` o resultado deverá ser o seguinte:

```
Loading projF1.lisp
Loading SolF1.lisp
Loading datastructures.lisp
Finished loading datastructures.lisp
Loading auxfuncs.lisp
Finished loading auxfuncs.lisp
Finished loading SolF1.lisp
Loading track: track0.txt
"Exercise 1.1 - isObstaclep"
  Solution is correct? T
  Solution is correct? NIL
"Exercise 1.2 - isGoalp"
  Solution is correct? T
  Solution is correct? NIL
"Exercise 1.3 - nextState"
  Solution is correct? T
  Solution is correct? NIL
Finished loading projF1.lisp
T
```

¹<https://mooshak.dcc.fc.up.pt/>

A implementação inicial no ficheiro inclui uma solução específica para uma dada entrada e não a implementação correcta. Daí não conseguir resolver todos os testes para os problemas 1.1, 1.2 e 1.3. Antes de tentar submeter a solução no Mooshak deverá implementar as funções pedidas de forma a que funcionem correctamente para todas as chamadas com valores válidos de acordo com com a especificação fornecida neste enunciado.

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

3 2a Fase (8 valores)

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>)
```

3.4 Procura não guiada (outros problemas) (2val)

Este teste irá verificar se as funções implementados são genéricas ou se o código foi feito à medida do problema ou da solução. A variável `problem` irá carregar um problema diferente e verificar se a solução encontrada é correcta ou não. O novo problema usa a mesma estrutura de dados para o estado mas as funções `isGoalp` and `nextStates` irão ser diferentes.

```
"Exercise 2.4 - iterlimdepthfirstsearch alternative unknown problem"
> (iterlimdepthfirstsearch <problem>
(<initial state> ... <goal state>)
```

3.5 Funções de teste

Deverão ser implementadas as funções, e tantas outras quantas necessárias, no ficheiro `SolF2.lisp`. A versão inicial deste ficheiro inclui já uma possível solução para a primeira fase do projecto. Ao executar o ficheiro `projF1.lisp`, já com a solução correcta o resultado deverá ser o seguinte:

```
Loading projF2.lisp
Loading SolF2.lisp
Loading datastructures.lisp
Finished loading datastructures.lisp
Loading auxfuncs.lisp
Finished loading auxfuncs.lisp
Finished loading SolF2.lisp
Loading track: track1.txt
"Exercise 2.1 - nextStates"
```

```

Solution is correct? T
"Exercise 2.1b - nextStates"
Solution is correct? T
"Exercise 2.2 - limdepthfirstsearch"
Solution is correct? T
"Exercise 2.3 - iterlimdepthfirstsearch"
Solution is correct? T
"Exercise 2.2b - limdepthfirstsearch"
Solution is correct? T
"Exercise 2.3b - iterlimdepthfirstsearch"
Solution is correct? T
Finished loading projF2.lisp
T

```

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

4 3a Fase (10 valores)

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

4.1 Cálculo de heuristica (3val)

Criar uma 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:

$$\begin{aligned} p^{t+1} &= p^t + a \\ p_l^{t+1} &= p_l^t + a_l \\ p_c^{t+1} &= p_c^t + a_c \end{aligned}$$

O resultado da função deverá ser lista de lista equivalente ao elemento `env` da estrutura `track` onde para cada localização 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> <M> <M> 10 9 8 7 6 5 4 3 2 <M> <M> <M> <M>)
(<M> <M> <M> <M> 10 9 8 7 6 5 4 3 2 1 <M> <M> <M>)
(<M> 14 13 12 11 10 9 <M> <M> <M> <M> 3 2 1 0 0 <M>)
(<M> 14 13 12 11 10 <M> <M> <M> <M> <M> 2 1 0 0 <M>)
(<M> 14 13 12 11 <M> <M> <M> <M> <M> <M> 1 0 0 <M>)
(<M> <M> <M>))
```

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>)
```

4.4 Funções de teste

Oportunamente serão publicados testes relativos à 3a fase do projecto.

5 Entregas, Prazos, Avaliação e Condições de Realização

5.1 Entregas e Prazos

A realização do projecto divide-se em 3 entregas.

As entregas serão feitas por via electrónica através do sistema Mooshak de acordo com instruções que serão oportunamente publicadas no site da cadeira.

Deverá ser submetido um ficheiro .lisp contendo o código do seu projecto. O ficheiro de código deve conter em comentário, na primeira linha, os números e os nomes dos alunos do grupo, bem como o número do grupo. Não é necessário incluir os ficheiros disponibilizados pelo corpo docente.

As entregas têm que ser feitas até ao limite definido a seguir, data e hora, não sendo aceites projectos fora de prazo sob pretexto algum²:

- 1^a Entrega - até às 23:59 do dia 21/10/2016
- 2^a Entrega - até às 23:59 do dia 11/11/2016
- 3^a Entrega - até às 23:59 do dia 09/12/2016

5.2 Avaliação

As várias entregas têm pesos diferentes no cálculo da nota do Projecto. A 1.^a entrega corresponde a 10% da nota final do projecto (ou seja 2 valores). A 2.^a entrega corresponde a 40% da nota final do projecto (ou seja 8 valores). Finalmente a 3.^a entrega, corresponde aos restantes 50% da nota final do projecto (ou seja 10 valores).

A avaliação da 1.^a e da 2.^a entrega será feita com base na execução correcta (ou não) das funções pedidas.

A avaliação da 3.^a entrega será feita com base na execução correcta (ou não) das funções pedidas e com base na qualidade dos resultados obtidos pelo programa na resolução de problemas VectorRacer.

5.3 Condições de realização

O código desenvolvido deve compilar em CLISP 2.49 sem qualquer "warning" ou erro. Todos os testes efectuados automaticamente, serão realizados com a versão compilada do vosso projecto.

Aconselhamos também os alunos a compilarem o código para os seus testes de comparações entre algoritmos/heurísticas, pois a versão compilada é consideravelmente mais rápida que a versão não compilada a correr em Common Lisp.

No seu ficheiro de código não devem ser utilizados caracteres acentuados ou qualquer carácter que não pertença à tabela ASCII, sob pena de falhar todos os testes automáticos. Isto inclui comentários e cadeias de caracteres.

É prática comum a escrita de mensagens para o ecrã, quando se está a implementar e a testar o código. Isto é ainda mais importante quando se estão a testar/comparar os algoritmos. No entanto, não se esqueçam de remover/comentar as mensagens escritas no ecrã na versão final do código entregue. Se não o fizerem, correm o risco dos testes automáticos falharem, e irão ter uma má nota na execução.

A avaliação da execução do código do projecto será feita automaticamente através do sistema Mooshak, usando vários testes configurados no sistema. O tempo de execução de cada teste está limitado, bem como a memória utilizada. Só poderá efectuar uma nova submissão pelo menos 15 minutos depois da submissão anterior. Só são permitidas 10 submissões em simultâneo no sistema, pelo que uma submissão poderá ser recusada se este limite for excedido. Nesse caso tente mais tarde.

Os testes considerados para efeitos de avaliação podem incluir ou não os exemplos disponibilizados, além de um conjunto de testes adicionais. O facto de um projecto completar com sucesso os exemplos fornecidos não implica, pois, que esse

²Note que o limite de 10 submissões simultâneas no sistema Mooshak implica que, caso haja um número elevado de tentativas de submissão sobre o prazo de entrega, alguns grupos poderão não conseguir submeter nessa altura e verem-se, por isso, impossibilitados de submeter o código dentro do prazo.

projecto esteja totalmente correcto, pois o conjunto de exemplos fornecido não é exaustivo. É da responsabilidade de cada grupo garantir que o código produzido está correcto.

Duas semanas antes do prazo da 1.^a entrega (isto é, na Segunda-feira, 7 de Outubro), serão publicadas na página da cadeira as instruções necessárias para a submissão do código no Mooshak. Apenas a partir dessa altura será possível a submissão por via electrónica. Até ao prazo de entrega poderá efectuar o número de entregas que desejar, sendo utilizada para efeitos de avaliação a última entrega efectuada. Deverão portanto verificar cuidadosamente que a última entrega realizada corresponde à versão do projecto que pretendem que seja avaliada. Não serão abertas excepções³.

Para submeter é necessário primeiro aceder ao link
<http://iashak.rnl.tecnico.ulisboa.pt/~mooshak/cgi-bin/getpass-ia>
e depois irão receber uma password para poderem submeter.

Atenção também que na submissão os includes a fazer tem de ter a extensão *.fas* e não a extensão *.lisp*.

Projectos muito semelhantes serão considerados cópia e rejeitados. A detecção de semelhanças entre projectos será realizada utilizando software especializado⁴ e caberá exclusivamente ao corpo docente a decisão do que considera ou não cópia. Em caso de cópia, todos os alunos envolvidos terão 0 no projecto e serão reprovados na cadeira.

5.4 Discussão dos projectos

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

6 Competição do Projecto

Vai ser realizada uma competição entre todos os projectos submetidos para determinar quais os melhores projectos a resolver problemas de VectorRacer. Para poderem participar na competição, a vossa implementação tem que passar todos os testes de execução referentes às funções pedidas no enunciado. A função a ser usada na competição é a função best-search. Os 3 melhores classificados na competição, ou seja os projectos que consigam a maior pontuação global a resolver o conjunto de problemas da competição, serão premiados com as seguintes bonificações:

- 1.^º Lugar: 1,5 valores de bonificação na nota final do projecto
- 2.^º Lugar: 1,0 valores de bonificação na nota final do projecto
- 3.^º Lugar: 0,5 valores de bonificação na nota final do projecto

References

- [1] *Common Lisp HyperSpec*, 2005.
- [2] *GNU CLISP - an ANSI Common Lisp Implementation*, 2010.
- [3] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, 3rd edition edition, 2010.

³Note que se efectuar uma submissão no Mooshak a menos de 15 minutos do prazo de entrega, fica impossibilitado de efectuar qualquer outra submissão posterior.

⁴Ver <http://theory.stanford.edu/~aiken/moss/>