Introdução à Inteligência Artificial

Licenciatura em Engenharia Informática, Engenharia Informática – Pós Laboral e Engenharia Informática – Curso Europeu

2º Ano – 1º semestre 2025/2026

Aulas Laboratoriais

Ficha 2: Agentes reativos

1. Percecionar a vizinhança de um agente

O NetLogo possui primitivas que permitem a um agente saber o que se passa na sua vizinhança e atuar sobre ela. Para isso o agente tem que, a partir da sua localização, saber aceder aos *patches* (ou células) do ambiente. Esse acesso é conseguido, no NetLogo, através de várias primitivas, usadas dentro da instrução ***ask turtles / ask turtle id***, das quais se apresentam os seguintes exemplos:

- ***patch-here*** – Célula onde o agente está, conforme se representa na figura 1.

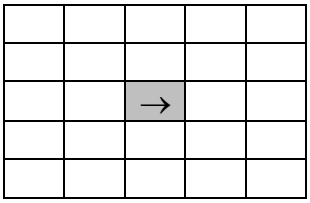

Figura 1 – Representação da célula tratada com a primitiva *patch-here*.

Exemplo 1 – analisar cor da célula e executar algo:

```
ask turtles[ ;; como só há um agente também pode ser ask turtle 0
      if [pcolor] of patch-here = red
      […]
]
```

Exemplo 2 – alterar a cor da patch:

```
ask turtles[ ;; como só há um agente também pode ser ask turtle 0
      ask patch-here
            [set pcolor red]
]
```

Esta vizinhança da célula atual, pode ser 'simplificada' fazendo uma análise direta à propriedade da *patch* (no exemplo, a propriedade *pcolor*) sem ser necessário referir a instrução `patch-here`. Em alternativa ao código acima poderia ser:

Exemplo 1

```
ask turtles[ ;; como só há um agente também pode ser ask turtle 0
      if pcolor = red
      […]
]
```

Exemplo 2:

```
ask turtles[ ;; como só há um agente também pode ser ask turtle 0
      set pcolor red
]
```

- ***patch-ahead distância*** – Se, por exemplo, distância for 1, será a célula vizinha da direita (localizada logo à direita da posição do agente), conforme se representa na figura 2. Se for diferente da unidade, será a célula correspondente à distância, em *patches*, a partir do agente, na mesma linha onde ele está (distância negativa vai para o lado esquerdo, enquanto que positiva vai para o lado direito).

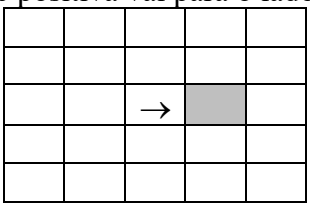

Figura 2 – Representação da célula tratada com a primitiva *patch-ahead*.

Exemplo 1 – ver a cor da patch e executar algo:

```
ask turtles[ ;; como só há um agente também pode ser ask turtle 0
      if [pcolor] of patch-ahead 1 = red
      […]
]
```

Exemplo 2 – alterar a cor da patch:

```
ask turtles[ ;; como só há um agente também pode ser ask turtle 0
      ask patch-ahead 1
            [set pcolor red]
]
```

- ***patch-left-and-ahead ângulo distância*** – Se, por exemplo, o ângulo for 45º e a distância for 1, será a célula que está acima à direita em relação à posição do agente, conforme se representa na figura 3. A distância funciona da mesma forma que a apresentada acima, enquanto o ângulo define a direção para onde se vai à procura do *patch*.

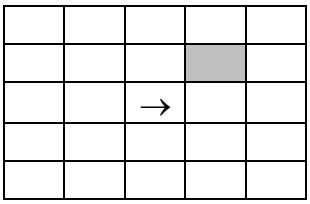

Figura 3 – Representação da célula tratada com a primitiva *patch-left-and-ahead*.

Exemplo 1 – ver a cor da patch e executar algo:

```
ask turtles[ ;; como só há um agente também pode ser ask turtle 0
      if [pcolor] of patch-left-and-ahead 45 1 = red
      […]
]
```

Exemplo 2 – alterar a cor da patch:

```
ask turtles[ ;; como só há um agente também pode ser ask turtle 0
      ask patch-left-and-ahead 45 1
            [set pcolor red]
]
```

- ***patch-right-and-ahead ângulo distância*** – Se, por exemplo, o ângulo for 45º e a distância for 1, será a célula que está abaixo à direita em relação à posição do agente, conforme se representa na figura 4. A distância e o ângulo funcionam como descrito acima.

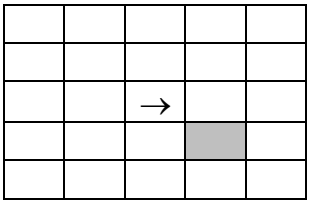

Figura 4 – Representação da célula tratada com a primitiva *patch-right-and-ahead*.

- ***neighbors4*** – Células que estão na vizinhança nas direções vertical e horizontal (localizadas logo à esquerda, à direita, acima e abaixo da posição do agente), conforme se representa na figura 5.

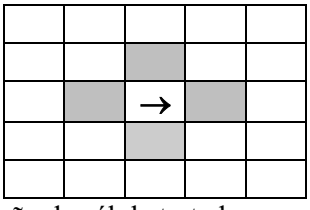

Figura 5 – Representação da célula tratada com a primitiva *neighbors4*.

- ***neighbors*** – Células que estão na vizinhança do agente, em qualquer direção, conforme se representa na figura 6.

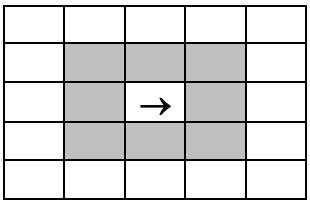

Figura 6 – Representação da célula tratada com a primitiva *neighbors*.

Estas vizinhanças permitem apenas uma perceção global das células e não uma análise individual. Isto é, é possível saber se nas 4 ou 8 células (patches) existe alguma de cor vermelha por exemplo, mas não qual delas é... Permite saber se existe algum agente nessas células, mas não em qual se posiciona. São funções úteis quando se pretende atuar simultaneamente sobre as patches, como se exemplifica abaixo.

Exemplo 1 – existe algum agente na vizinhança?

```
ask turtles[ ;; como só há um agente também pode ser ask turtle 0
      if any? turtles-on neighbors  […]
]
```

Exemplo 2 – atuar sobre a vizinhança alterando a cor:

```
ask turtles[ ;; como só há um agente também pode ser ask turtle 0
      ask neighbors [set pcolor blue]
]
```

Para um melhor entendimento do conceito de um agente saber o que se passa na sua vizinhança e atuar sobre ela, apresenta-se a primeira parte de um desafio:

a) Abra o modelo guardado no ficheiro ***IIA_Ficha2_Exercicio1.nlogo***, que está disponível no *Moodle*. Neste modelo é criado apenas um agente (identificado por uma seta), com a capacidade de poder alterar a cor das *patches* imediatamente vizinhas, quando quiser. Em cada iteração, esse agente pode deslocar-se na direção para que a seta aponta ou rodar 90º na direção dos ponteiros do relógio. O modelo já vem com todo o interface criado, i.e., com os botões para a sua inicialização (***Setup***), o deslocamento do agente (***Go***), a rotação do agente (***Turn***) e a reposição da cor inicial do ambiente (***ResetColor***). Tem, ainda, um *chooser*, que indicará quais serão as células vizinhas ao agente que deverão ser pintadas a outra cor, quando pressionado o botão ***Paint***;
b) Consulte o código para compreender exatamente o objetivo do modelo;
c) Termine a implementação da função ***Paint*** que altera a cor das *patches* vizinhas do agente. Chama-se a atenção para uma das hipóteses existentes no *chooser*, designada por ***Forward_3***, que necessita de usar em conjunto as primitivas ***patch-ahead***, ***patch-left-and-ahead*** e ***patch-right-and-ahead***. Na implementação, use o dicionário do Netlogo se necessitar de ajuda e tenha em consideração que o agente possui uma direção de deslocamento.

Uma vez consolidado o conhecimento anterior, pode-se passar à segunda parte do desafio:

a) Abra o ficheiro ***IIA_Ficha2_Exercicio2.nlogo***, que está disponível no *Moodle*. Neste modelo o agente continua a ter a capacidade de alterar as células na sua vizinhança. As diferenças em relação ao exemplo anterior são:
   - O agente não tem uma direção de movimentação explícita. Pode deslocar-se para cima, para baixo, para a esquerda ou para a direita, sem ser necessário efetuar movimentos de rotação, bastando para isso usar os respetivos botões;
   - O agente considera sempre a mesma vizinhança de 8 células (baseada na primitiva *neighbors*), também conhecida como vizinhança de *Moore*;
   - O ambiente inicial é constituído por células de duas cores.

   O agente desloca-se pelo ambiente, tendo como objetivo alterar a cor de todas as células brancas para preto. Sempre que uma célula branca aparece na sua vizinhança o agente pode alterar a sua cor. Em cada iteração só pode ser alterada a cor de uma *patch*. Se, numa determinada iteração, existirem várias *patches* brancas na vizinhança, o agente deve escolher uma delas aleatoriamente para efetuar a modificação. O contador pontos que existe no modelo vai contabilizando as células que mudam de cor.
b) Consulte o código para compreender exatamente o objetivo do modelo;
c) Implemente o procedimento ***Verify*** que verifica se existem uma ou mais células com cor branca na vizinhança do agente. Se existirem deve mudar a cor de uma delas para preto e incrementar a variável global *scores*. Para realizar esta tarefa pode necessitar de utilizar as primitivas ***any?*** e ***one-of*** (explore-as no dicionário do Netlogo).

2 – Agentes Reativos

O exercício anterior permitiu saber usar as funções Netlogo para percecionar as células vizinhas de um agente (*turtle*). O que se pretende agora é utilizar essas funções para implementar um modelo com agentes reativos.
Para criar um modelo de simulação deste tipo é necessário escrever um conjunto de funções (*procedures*) que agrupam vários comandos do NetLogo e que deverão permitir:

- Criar o ambiente;
- Definir parâmetros;
- Controlar o comportamento dos agentes;
- Indicar os meios de apresentação de resultados.

Assim, imagine-se um ambiente com dois tipos de agentes reativos: as formigas e os caracóis. Estes agentes deambulam no seu mundo até encontrarem o respetivo "ninho" ou morrerem. Esse mundo possui armadilhas (*patches* vermelhas) e dois "ninhos" (a célula azul é o ninho das formigas e a célula amarela é o ninho dos caracóis).
Os agentes deverão possuir **perceções** diferentes.

- As formigas deverão conseguir ver o que está na célula à sua frente
- Os caracóis só poderão ter conhecimento da célula que ocupam.

Um exemplo de representação deste modelo pode ser visto na figura 7. De forma mais detalhada, os comportamentos dos agentes deverão ser tratados da seguinte maneira:

- **Formigas**
  - Se na célula à frente estiver uma armadilha, roda à direita 90º;
  - Se na célula à frente estiver o ninho, avança e "desaparece" dentro dele (use o comando ***die***, na próxima aula este comportamento será alterado);
  - Se não for nenhuma das situações anteriores, a formiga avança em 90% das vezes, roda à direita em 5% das vezes e roda à esquerda nas restantes 5% das vezes.

- **Caracóis**
  - Se na célula em que está estiver uma armadilha, morre;
  - Se na célula em que está estiver o ninho, desaparece dentro dele (use o comando ***die***, na próxima aula este comportamento será alterado);
  - Se não for nenhuma das situações anteriores, o caracol avança;

O modelo deverá ainda ter as seguintes características adicionais:

- As armadilhas mantêm-se inalteradas ao longo de toda a simulação;
- A simulação termina quando não existirem agentes a habitar o mundo (todos os agentes estão no respetivo ninho ou mortos);
- A simulação do comportamento probabilístico dos agentes e do mundo deverá ser baseado na função ***random***.

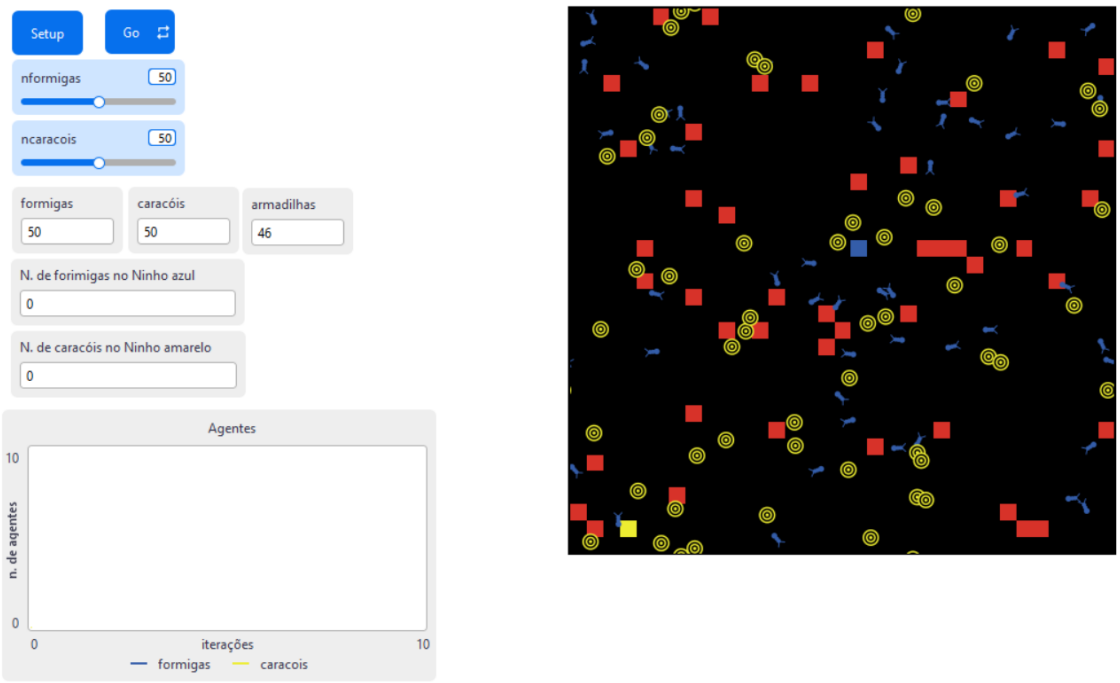

Figura 7 – Modelo com agentes reativos.

Para implementar este modelo de agentes reativos seguir os seguintes passos:

a) Começar um novo modelo, usando a opção ***File → New***, do menu principal e ir gravando o ficheiro enquanto se vai implementando o modelo;
b) Criar dois botões e associar a um deles o nome do procedimento ***Setup*** e ao outro botão (do tipo *forever*) o procedimento ***Go***;
c) No separador CODE, criar dois tipos de *turtles* com os seguintes comandos:

```
breed[formigas formiga]
breed[caracois caracol]
```

d) Criar o procedimento ***Setup*** com os seguintes comandos:

```
to Setup
  setup-patches
  setup-turtles
end
```

e) Criar o procedimento ***setup-patches***, implementando as seguintes funcionalidades:
   i. Limpar todas as variáveis, *patches*, agentes, etc. (usar o comando ***clear-all***);
   ii. Alterar o tamanho das *patches* para 15 pixels (usar o comando ***set-patch-size 15***);
   iii. Colocar as armadilhas em 5% de todo o ambiente, representadas por *patches* de cor vermelha (ver como foi colocada a comida no exercício da ficha 1);
   iv. Criar o ninho das formigas (uma *patch* de cor azul) e o ninho dos caracóis (uma *patch* de cor amarela), em posições aleatórias (usar os comandos ***ask one-of patches with [pcolor = black]***).

f) Criar o procedimento ***setup-turtles***, implementando as seguintes funcionalidades:
   i. Criar as variáveis ***nformigas*** e ***ncaracois***, associadas a dois *sliders*, no separador *Interface*;
   ii. Criar o número de agentes formigas e agentes caracóis definido nas variáveis acima (ver como foram criados os homens e as mulheres no exercício da ficha 1);
   iii. As formigas deverão ter a forma do tipo "***bug***", cor azul, coordenadas aleatórias (usar os comandos ***setxy random-xcor random-ycor***) e com orientação inicial de 90º. É preciso garantir que não ficam posicionadas em cima de uma armadilha (para fazer isso, definir uma posição aleatória e entrar num ciclo ***while*** que define nova posição aleatória enquanto a cor do *patch* dessa posição for diferente de preto);
   iv. Os caracóis deverão ter a forma do tipo "***target***", cor amarela e coordenadas aleatórias. É preciso garantir que não ficam posicionadas em cima de uma armadilha.

g) Criar o procedimento ***Go*** com os seguintes comandos:

```
to Go
   MoveFormigas
   MoveCaracois
   if count turtles = 0 [stop]
end
```

h) Criar o procedimento ***MoveFormigas***, implementando as seguintes funcionalidades:
   i. Verificar a cor da *patch* logo à sua frente;
   ii. Se for vermelha, rodar à direita 90º (usar o comando ***right graus*** ou ***rt graus***);
   iii. Se for azul, avançar uma posição (usar o comando ***forward distância***) e desaparecer (usar o comando ***die***);
   iv. Se não for nenhuma das situações anteriores, avançar uma posição em 90% das vezes, rodar à direita em 5% das vezes e rodar à esquerda nas restantes 5% das vezes (esta implementação é semelhante à colocação das armadilhas).

i) Criar o procedimento ***MoveCaracois***, implementando as seguintes funcionalidades:
   i. Verificar a cor da *patch* que habita;
   ii. Se for vermelha, morrer;
   iii. Se for amarela, desaparecer;
   iv. Se não for nenhuma das situações anteriores, avançar uma posição.

j) Testar os procedimentos ***Setup*** e ***Go***, corrigindo eventuais erros;

k) No separador *Interface* acrescentar três elementos do tipo *monitor*, que permitam apresentar o número de formigas, o número de caracóis e o número de armadilhas presentes no ambiente (usar o comando ***count*** no código do respetivo *monitor*);

l) Acrescentar mais dois elementos do tipo *monitor*, que permitam mostrar quantas formigas e quantos caracóis chegam ao respetivo ninho. Para isso:
   i. Criar duas variáveis globais, colocando depois dos comandos ***breed*** a seguinte instrução:
      globals [nNinhoAzul nNinhoAmarelo]
   ii. Inicializar essas variáveis no procedimento ***Setup-patches*** através do comando:
      set nNinhoAzul 0
      set nNinhoAmarelo 0
   iii. Sempre que um dos agentes chegar ao seu respetivo ninho, atualizar a variável correspondente, usando um dos seguintes comandos:
      set nNinhoAzul nNinhoAzul + 1
      set nNinhoAmarelo nNinhoAmarelo + 1

m) No separador *Interface* acrescentar um elemento do tipo *plot*, de nome Agentes, que possua duas canetas, uma azul para o número de formigas e uma amarela para o número de caracóis. Para a caneta das formigas, usar a instrução ***plot count formigas*** e para a caneta dos caracóis, usar a instrução ***plot count caracóis***;

n) No procedimento ***Setup***, inclua a instrução ***reset-ticks***, conforme apresentado abaixo:

```
to setup
  setup-patches
  setup-turtles
  reset-ticks
end
```

o) No procedimento ***Go***, inclua a instrução ***tick***, para atualizar o gráfico, conforme apresentado abaixo:

```
to Go
  moveFormigas
  moveCaracois
  if count turtles = 0 [stop]
  tick
end
```

p) No separador *Interface* testar as funcionalidades acrescentadas.

O modelo criado e simulado até aqui pode ser melhorado e incrementado com novas características interessantes. Assim, deixam-se os seguintes desafios:

- **Alterar posição das armadilhas**
  Criar o procedimento ***changeArmadilhas***, que deverá alterar a posição das armadilhas, mantendo sempre o seu número inicial. É importante ter-se o cuidado de não destruir os dois ninhos, nem colocar as armadilhas nas *patches* onde estão os agentes. Este procedimento deve ser chamado no procedimento ***Go***. De maneira a que o modelo fique mais completo, introduza um elemento do tipo *switch* que permita alterar ou não a posição das armadilhas aquando da simulação;

- **Alterar as perceções dos caracóis**
  Criar o procedimento ***MoveCaracois2***, implementando as seguintes funcionalidades:
  i. Verificar a cor da *patch* logo à frente, à esquerda (90º) e à direita (90º);
  ii. Se o ninho estiver numa dessas três células, avançar para ele e desaparecer;
  iii. Se numa dessas três células estiver uma armadilha, evitá-la, rodando ou avançando;
  iv. Se não for nenhuma das situações anteriores, avançar uma posição em 90% das vezes, rodar à esquerda em 5% das vezes e rodar à direita nas restantes 5% das vezes.

  De maneira a que o modelo fique mais completo, introduza um novo elemento do tipo *switch* que permita alterar ou não o comportamento dos caracóis aquando da simulação;

- **Mimetismo**
  Criar um procedimento que simule o fenómeno do mimetismo. Ou seja, se agentes de dois tipos se encontrarem em patches vizinhas, a formiga assume a cor do caracol. Assim que os agentes se afastem (i.e., assim que deixarem de ser vizinhos), retomam as cores originais. Este procedimento deve ser chamado em ***Go***. De maneira a que o modelo fique mais completo, introduza um novo elemento do tipo *switch* que permita aplicar o mimetismo ou não aquando da simulação.