**Introduction to Artificial Intelligence**

Bachelor's in informatics Engineer and Informatics Engineer – European course

2º Year – 1º semester

Practical classes

# Assignment 2: Reactive Agents

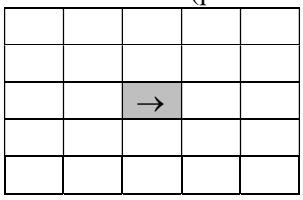
## I – Perceptions

Some NetLogo primitive features allow an agent to know what goes on in its neighborhood and act on it. In the following two examples an agent moves for the environment and can change the colors of the surrounding patches.

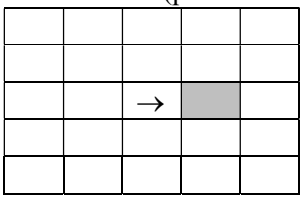
### 1. Assignment 1

In this example the aim is to develop a simple model in which an agent (indicated by arrow) can change the color of the surrounding patches. This model considers 5 types of neighborhood:

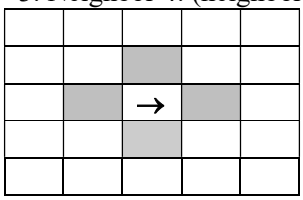
1. Current cell (patch-here)



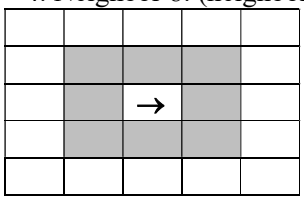
2. Forward (patch-ahead)



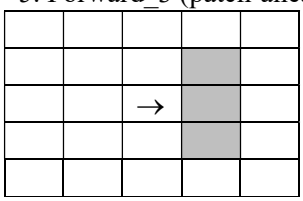
3. Neighbor 4: (neighbors4)



4. Neighbor 8: (neighbors)



5. Forward_3 (patch-ahead. patch-left-and-ahead, patch-right-and-ahead)



In the model that is available in Moodle (**ex1.nlogo**) an environment is created with only one agent. At each iteration, the agent can move forward or turn left.

If the **Paint** button is selected, the agent changes the color of some patches (according to the neighborhood that is indicated on the **chooser** to the right of this button).

## Tasks to perform

**1.** Open **IIA_Ficha2_Exercicio1_inicio.nlogo** file and see the code to clearly understand the purpose of the model.

**2.** Complete the implementation of the **Paint** function that changes the color of the agent's neighboring patches.

**To accomplish the second task may need to use some of the following primitives (explore them in Netlogo dictionary): patch-ahead, patch-left-and-ahead, patchright-and-ahead, patch-at, neighbors, neighbors4.**

**2. Assignment 2**

In this example (model **IIA_Ficha2_Exercicio2_inicio.nlogo**) the agent continues to have the ability to change the cells in their neighborhood. The differences from the previous example are:

- The agent has no explicit direction of movement. May move up, down, left or right without requiring rotation movements.
- The agent always considers the same neighborhood of 8 cells (Neighbor_8 the previous example, also known as Moore's neighborhood).
- Two color cells constitute the initial environment.

The agent moves in the environment, aiming to change the color of all white cells to black. Whenever a white cell appears in your neighborhood the agent can change its color. In each iteration only the color of a patch can be changed. If, in a given iteration, there are several white patches in the neighborhood, the agent must choose one at random to perform the modification.

The counter scores how many cells is changing color.

## Tasks to perform

**1.** Open **IIA_Ficha2_Exercicio2_inicio.nlogo** file and see the code to clearly understand the purpose of the model.

**2.** Implement the **Verify** function that checks for one or more cells with white on the agent's neighborhood. If there is/are any, the color of one of them must be changed to black and the variable ***scores*** must be increased.

**To accomplish the second task may need to use some of the following primitives (explore them in Netlogo dictionary): any?, count, one-of .**

## II – Reactive Agents

The goal of this exercise is to model two types of reactive agents: **ants** and **snails**. The agents move in the environment until they find their nest, trying to avoid the pitfalls in the environment (red cells).

### Environment:

** 5% of red cells – traps
** one yellow cell – snails' nest
** one blue cell – ants' nest

### Agents

Perceptions

- Ants can see what kind of cell is in the patch ahead.
- Snails can see what kind of cell is in the current patch (patch here)

Actions

- **Ants** can move forward, turn left or right (at each iteration, only one action must be performed). If a trap is detected in the analyzed cell, the best action to avoid it must be taken. If the blue nest is detected in the analyzed cell, the ant must go to the nest and rest there. When the agents are in the 'black area', they must move forward with 90% of probability or turn left (5% of probability) or right (5% of probability).

- **Snails** can move forward (at each iteration, only one action must be performed).

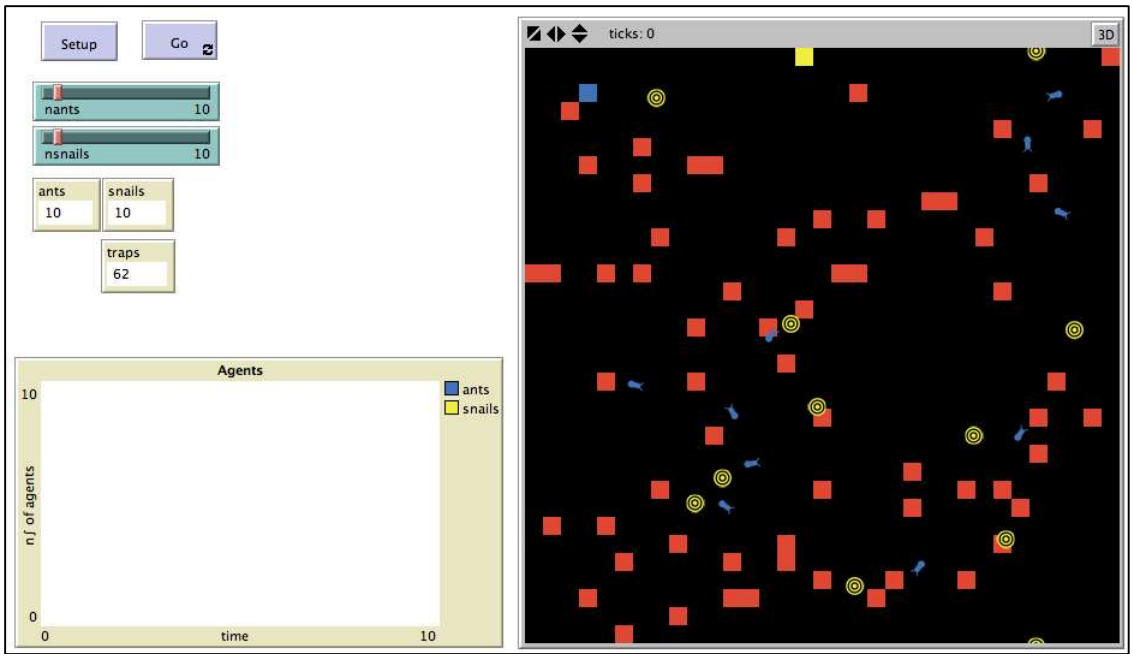
  If a trap is detected in the analyzed cell, the snail dies, because it fell in the trap. If the yellow nest is detected in the analyzed cell, the snail must rest there. When the agents are in the 'black area', they must move forward, because it's the only action they can perform.

Features

- **Ants** must be blue, with shape "bug"
- **Snails** must be yellow with shape "target"
- Both agents must be placed at random coordinates, using
  ```
  setxy random-xcor random-ycor
  ```
- The number of ants and snails to create must be configured in the environment using two *sliders*.

The simulations stops when all agents are deed or in their nests.

Next Figure shows the interface you must build and in the environment are the traps and the agents created after pressing the Setup button.




To implement these reactive agents take the following steps:

## Step 1 – Buttons Setup e Go

a) Start a new model using the option **File - New** from the menu and save the file.
b) Create two buttons in the interface. Fist button will call procedure **setup** and the second button (check ***Forever*** option) will start the movement of the agents (procedure **go**)
c) Create two sliders in the environment. Each slider must have a variable name that will be used to create the correct number of ants and snails. You can use, as in the figure, **nants** and **nsnails**.

## Step 2 – Code: Breed, Setup e Go

a) In the tab CODE create two kind of *turtles* using the primitve *breed*:

```
breed[ants  ant]
breed[snails snail]
```

b) After that write the procedure ***setup*** as follows. This procedure is associated to the first button and will call two new procedures, one to configure the environment (**`setup-patches`**), and other to create and setup the agents (**`setup-turtles`**). Those two procedures, will be made next.

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

c) Write the procedure ***setup-patches*** with these tasks:
   a. Clear all *patches*, variables and turtles
   b. Change the patch size to 15 px (`set-patch-size 15`)
   c. Change 5% of the patches to red
   d. Choose one the black cells and create the blue nest (change the patch color to blue)
   e. Choose one the black cells and create the yellow nest (change the patch color to yellow)

d) Write the procedure ***setup-turtles*** with these tasks:
   a. Clear all turtles
   b. Create the correct number of ants and snails. Use the variables `nants` and `nsnails` created in the two sliders of the INTERFACE.
   c. Configure the ants and snails with the features described above.
   d. Assure that they don't are placed in a trap (use a **while** instruction to change their positions if this happens)

e) Write the procedure ***go***:

```
to go
  move-ants
  move-snails
  if count turtles = 0  ;;count turtles -> conta o nº de agentes
   [stop]
end
```

f) Write the procedure ***move-ants*** with the movement of the ants. Use a set of ***ifelse*** instructions to program the actions described on page 3.

**Example:**

```
ask ants[
    ifelse [pcolor] of patch-ahead 1 = red
            [right 90]
            [ifelse [pcolor] of patch-ahead 1 = blue
                    […]
                    […]
            ]
]
```

g) Write the procedure ***move-snails*** with the movement of the snails. Use a set of ***ifelse*** instructions to program the actions described on page 3.

**Example:**

```
ask snails[
    ifelse pcolor = red
            [die]
            [ifelse pcolor = yellow
                    [die]
                    […]
            ]
]
```

h) Go to the interface and test the buttons setup and go.

## Step 3: Monitors

a) In INTERFACE add 3 *monitors* to present the number of **ants**, **snails,** and traps. Use the primitive `count` in each monitor) Example `count snails`

b) Add two additional *monitors* to count how many snails and ants arrived to the nests. To perform this task:
   a. In CODE create two global variables after the breed command:
      ```
      globals [blue-nest yellow-nest]
      ```
   b. Go to the procedure `setup-patches` and initialize these variables:
      ```
      set blue-nest 0
      …
      ```
   c. Go to the procedure `move-ants` and `move-snails` and when an agent reaches the nest, increase the corresponding variable:
      ```
      set blue-nest blue-nest + 1
      …
      ```

## Step 4: Plots

a) Go to INTERFACE and add a **Plot**. The name of the plot must be ***Agents*** and you must create two pens: the blue pen to count the number of ants and the yellow pen to count the number of snails. Example of the first pen: `plot count ants`

b) In order to update the plot at each iteration, go to the procedure ***setup*** and add the command `reset-ticks:`

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

c) And in the procedure ***go***, add the command `tick`

```
to go
  move-ants
  move-snails
  if count turtles = 0
   [stop]
  tick
end
```

d) Go to INTERFACE and test the model.

## Try to implement other ideas to change this model

a) Create a new procedure to change the positions of the traps. The same number must be kept, but their positions change at each iteration. Be careful to don't destroy the nests. The call to this new procedure must be performed in the procedure ***go***.

b) Make the snails a little bit more rational…
Create a new procedure `move-snails-2`. Where the snails can see the type of cell ahead, at its left and at its right. This way, they can avoid the traps. Include the ability

of turning left or right where a snail is in the 'black area'. Se the probabilities you consider the best. Call this procedure in `go`, instead of `move-snails`

c) Write a new procedure that simulates the idea of mimetism: if agents of different breeds are in neighbor patches, the ant becomes yellow.
When agents are no longer in neighbor cells, the ants turn blue again.
Use the commands **neighbors**, **snails-on**, **ants-on**, …). Call this procedure in **go**