3 - Agents with memory

In this assignment, memory will be added to the agents. According to the state of the memory, some behavior will be changed:

- **Ants and snails can reproduce**
  If their energy reaches a minimum value, that can reproduce with a certain probability;
- **Ants can eat snails**
  An ant can eat a snail, if has eaten at least 10 units of grass. After eating a snail, the grass counter is set to zero again;
- **Ants have a limited food storage capacity**
  An ant can only store a limited amount of grass. If the maximum storage capacity is reached (e.g. 10 units), the ant cannot eat more. At this point, they must find their nest (blue cell) and dump the stored food. If the nest is reached before attaining the maximum capacity, the ant can put in the nest what have stored so far.

So, make the following changes in the code of last assignment (or in the file ***IIA_Class3_Start.nlogo*** in the *Moodle*):

Step 1: When a nest is reached agents don't die

- Go to the procedures ***move-ants*** and ***move-snails*** and remove the command ***die*** when the nest is reached.

Step 2: Create food (grass) in the environment

- Change the procedure ***setup-patches*** in order to maintain the 5% red patches and adding 15% of green cells. The nests must be kept as they are.

Step 3: Agents can eat

- Add a new property called ***energy*** to all agents, ***turtles-own [energy]***;
- Go to procedure **setup-turtles** and start the energy of the agents with the value 100;
- Change the ***move-ants*** procedure, adding a new perception/action (a new ***ifelse*** command, which must be the first, since eating is now the most important action to the agents). If a green patch is in front of the ant, move forward, eat grass (patch becomes black), and increase ant's energy with 50. In all other movements of the ant, one unit of energy must be decreased;
- Change the ***move-snails*** procedure, adding a new perception/action (a new ***ifelse*** command, which must be the first, since eating is now the most

important action to the agents).). If a green patch is under the snail, eat grass (patch becomes black), and increase ant's energy with 50. In all other movements of the ant, one unit of energy must be decreased.

Step 4: Agents can die

a) Complete procedure ***go***, calling the new procedure ***check-death***:

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

b) Write the procedure ***check-death***:
   1. If the turtle's energy is lesser or equal than zero, the agent dies (primitive ***die***).

c) Run the model and verify if there are some errors.

Step 5: Add memory to ants

- Ants have a counter that measure how many units of grass have eaten. After eating at least 10 units of grass, if a snail is in the ant's perception neighborhood, the ant can eat the snail, absorbing its energy and killing it. After that, the grass counter must be set to zero. So,
  a) Create a new property to ants (memory), ***ants-own [grass-counter]***;
  b) Go to procedure ***setup-turtles*** and set the initial value of the counter to zero;
  c) Go to procedure ***move-ants*** and update the ***grass-counter***, each time an ant eats an unit of grass;
  d) Add a new perception/action to the procedure ***move-ants***:
     1. If there is a snail in the patch ahead and the ***grass-counter > 10*** the ant gets the snail's energy, the snail dies and the ***grass-counter*** is set to zero;
     2. Otherwise, the previous behavior is kept as it is.

Step 6: Ants' memory has limited capacity

a) Add a new *slider* in the *interface* to control the maximum storage capacity of the ants (***max-grass***);
b) If the ***grass-counter >= max-grass*** the ant cannot eat any more. Change this in the procedure ***move-ants***;
c) When the nest is achieved, update the variable ***blue-nest***, adding the value of ***grass-counter***. After that, ***grass-counter*** must be set to zero.

Step 7: Reproduction

a) Go to *Interface* and add a *switch* to activate/deactivate the agent's reproduction. Use variable ***reproduce?***;
b) Add three *sliders* to choose the reproduction probability and the energy threshold that is necessary for reproduction. Use the following variables:
   1. ***reproduction-ants*** – values from 0 to 100;
   2. ***reproduction-snails*** – values from 0 to 100;
   3. ***birth-energy*** – values from 100 to 500.

c) In the procedure ***go***, add the call to a new procedure called ***reproduction***:

```
to go
   move-ants
   move-snails
   check-death
   if reproduce? [reproduction]
   if count turtles = 0 [stop]
end
```

d) Write the procedure **reproduction**:
   1. If the ant's energy is higher than ***birth-energy***, use the probability chosen in ***reproduction-ants*** to reproduce the ant:
      i. Divide the ant's energy by 2;
      ii. Create a new ant (use **hatch 1**);
      iii. The new ant must be placed 5 patches ahead its parent (use **hatch 1 [jump 5]**);
      iv. Make the similar code for the snail's reproduction.

e) Go to *Interface* and test the model.

Step 8: Regrowth of grass

a) In the procedure ***go***, add the call to a new procedure called ***regrow-food***:

```
to go
   move-ants
   move-snails
   check-death
   if reproduce? [reproduction]
   regrow-food
   if count turtles = 0 [stop]
end
```

b) Write the procedure regrow-food:
   1. If the number of green patches is lesser than 50, use a probability of 2% to change some black cells to green.

Step 9: Adding a label to the agents

a) Go to *Interface* and add a new *switch* ***show-energy?***;

b) Go to the procedures ***setup-turtles*** and ***go*** and call a new procedure ***display-labels***;

c) Write the procedure ***display-labels***:

```
to display-labels
   ask turtles[
      set label ""
      if show-energy? [set label energy]
   ]
end
```

d) Go to *Interface* and check the model's functionality.

Step 10: Add competition

a) Write a new procedure called ***competition*** that mimics the competition between the two types of agents:
   1. If an ant and a snail are in the same neighborhood, the agent with the highest energy wins the competition, kills the enemy and gets its energy. Call this new procedure in **go**.