The GAML language

<u>Implementation of simple models</u>

<u>Exercice 1 - *Opinion Diffusion model [Deffuant et al.]*</u>

<u>Exercice 1.1: Simple version</u>
Aim:
The aim of this aspatial model is to simulate the opinion diffusion in a population of agents.

Description:
An opinion is represented by a numerical float value between 0.0 and 1.0 (initialised with a random value). A each step, agents choose a random other agent and update their opinions as follows.
If agents x and y update their opinions, their opinions will become:

- $\text{Opinion}_x' = \text{opinion}_x + \text{speedConvergence} * (\text{opinion}_y - \text{opinion}_x)$
- $\text{Opinion}_y' = \text{opinion}_y + \text{speedConvergence} * (\text{opinion}_x - \text{opinion}_y)$

Inputs:
The parameters of the simulation are the number of agents and the speed of convergence (*speedConvergence*) that is a number in [0.0,1.0].

Output:
Plot the opinion of each agent. What do you observe?

<u>Exercice 1.2: Improvement</u>
Agents with opinions very far will not influence each other's. To reproduce this, add a new parameter *threshold* representing the threshold of influence: if the difference of opinions between the agents is greater than this threshold they will not influence each others.
What do you observe on the plot?

Exercice 2 - *RumorMill model [From Netlogo model library]*

Aim:
The aim of this model is to simulate the spread of a rumor among a population of agents.

Description:
We consider a set of nb_agents people agents organized on a grid (a grid of dimension x dimension cells, dimension being a model parameter).
Initially *nb_init* agents know a rumor.
At each step, people agents who are aware of the rumor will say it to one of their neighbors.

Inputs:
As parameters of the simulation, we will add:
- The number of initial people having heard the rumor (*nb_init)*
- The number of agents : the height and width *dimension* will be defined as a parameter

Outputs:
Draw 3 displays
- 1 with agents aware of the rumor in red and agents that are not aware in white.
- 1 drawing agents with a "gradient" of color depending on the number of times the agent heard the rumor.
- 1 drawing agents with a "gradient" of color depending on the first time they heard the rumor.

Draw 3 plots:
- 1 display with the ratio of aware people agents;
- 1 display representing the acceleration of the rumour spread: it plots the ratio of the number of aware people in the current step over the number of aware people at the previous step.
- 1 display with the successive differences of the number of aware people in the current step over the number of aware people at the previous step.

HINT:

One way to model the problem.
Description:
People agents are characterized by three attributes:
- timesHeard: it tracks the number of times the rumor has been heard
  type: integer ; initial value: 0
- firstHeard: cycle when first heard the rumor
  type: integer ; initial value: -1
- hasJustHeard: tracks whether rumor was heard this round
  type: Boolean ; initial value: false

In addition they will have three attributes of color for each of the displays.

Dynamics of agents:
The people agents have the 2 following behaviors (reflex):

- spreadRumor: If the agent has already heard the rumor, it chooses randomly 1 neighbor agents and tells it the rumor, i.e. the hasJustHeard attribute is set at true.
- Update: If the agent has just heard the rumor, hasJustHeard is reset at false. Then firstHeard, timesHeard and the colors are updated if necessary.

Description of the global agents:
Two additional attributes are added to the global to store the number of aware agents in the current cycle and at the previous cycle.
An additional reflex is added to save the number of unaware people in the previous cycle and compute the number of aware people at the current cycle.

Exercice 3 - *Scheduling on the RumorMill model*

In the previous hint, the agents are updated (e.g. color…) once a step. But it can be possible that they have been infected by the rumor in a given step and be updated only at the following step (because they have behaved before being infected by the rumor in the current step).

The objective of this exercise it to tackle this issue. A classical way to deal with this issue is to drive the simulation from the global.

1. In people agents : replace the reflexes by actions (that can be called from the global)
2. In the global, add a reflex that
    a. Asks to agents aware of the rumor to spread it
    b. Asks to agents that have been told the rumor, to update their attributes
    c. Computes the number of people aware in the previous step and at this step.

Exercice 4 - Scheduling

This exercise aims at showing the effect of the scheduling of agents on results.

Write a model that:

- Define a *people* species with a *money* attribute.
- At each step, a global reflex fills a list of N random numbers, where N is the number of agents.
- The behaviour of each agent is to choose the greater value in the list, to add it to its money and to remove it from the list.

Do not hesitate to write useful debugging information to check the correctness of the agents' behaviour.

Plot the difference of money between the maximum and minimum values of money among the agents.

By default, agents are scheduled in the same order at each step (which can be responsible of the continuous). Modify this scheduling order to limit the difference of money between agents (see *schedules* facet on the species people):

- Execute agents in a random order.
- Execute agents sorted by their money attribute (in ascending order).

Exercice 5 - Containers and container-related operators

For the following questions, you need only a very simple model with the global, its init block and an empty experiment. All the answers can be written in the global init. Use the *write* statement to display and check your answers.
It is strongly advised to do the exercises in the provided order: every time a new GAML operator (or keyword) is needed, it is introduced with a hint, but only once.

All these questions can be answered in 1 line.

1. Declare and create a list of 10 chosen integer numbers in [0,10].
   (hints: variable declaration, create a list)
2. Get the number of elements of the list.
   (hint: length operator)
3. Get the first element of the list.
   (hint: first operator)
4. Reverse the order of the elements of the list.
   (hint: reverse operator)
5. Get the last element of the list.
6. Get the last element of the list.
   (hint: last operator)
7. Sort the element by ascending order.
   (hint: sort_by operator)
8. Sort the element by descending order.
9. Get a new list with only the elements greater or equals to 5.
   (hint: where operator)
10. Get the list of elements organised as follow: first the elements lower than 5, ordered by ascending order and then the elements greater or equals to 5, ordered by descending order.
    (hint: + operator to concatenate lists)
11. Get the list of all the elements multiplied by 10
    (hint: accumulate/collect operator)
12. Get the list of the square of elements greater than 5.

Define a species people, with 2 attributes: energy and money (random float values between 0.0 and 10.0).
Create 10 agents people.
As previously, write in 1 line:

1. Get the list of people with energy greater than 5.
2. Get a list of 3 random people agents.
   (hint: among operator)
3. Get the people agent with the max of energy, with the min of money.
   (hint: with_min_of, with_max_of operators)
4. Get the maximum value of money, the minimum value of energy among the people agents.
   (hint: min_of, max_of operators)
5. Get the list of all the energy values.
6. Get the list of people sorted by money.
7. Get the list of people with energy greater than 3 and lower than 6.

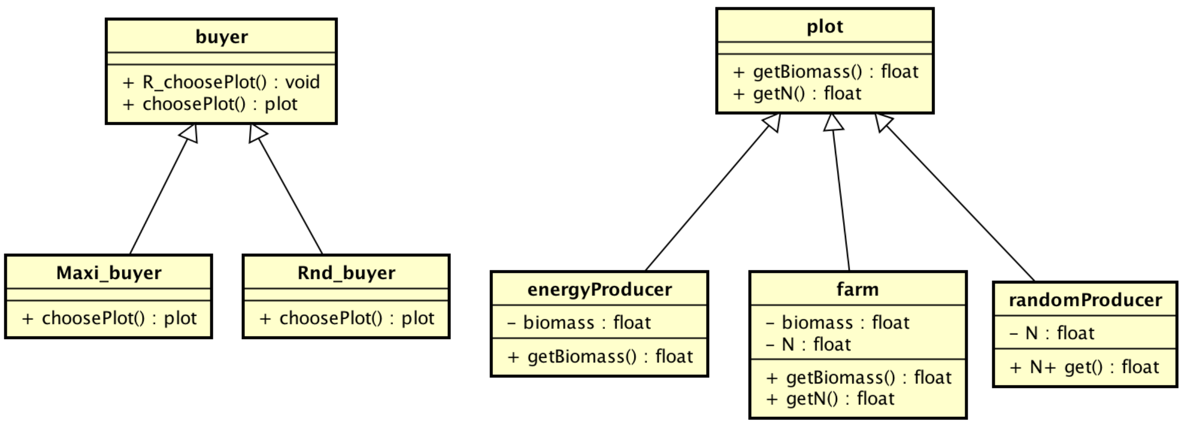
8. Get the people agent with the lowest money among people agents with energy greater than 2.
9. Check (i.e. get true of false) whether there is a people agent with money greater than 9.
   (hint: empty operator)

Define a map variable containing elements of type string (for the key) and float (for the value), initialize it at hand with 3 pairs (with at least 1 key starting with “a” letter).

1. Get the list of all the values of the map.
2. Get the map of pairs where the key start with the letter “a”
3. Given people agents of the previous question, get the map of people name and people nergy.
4. Get the map associating the name of the people agent, with the list of its 3 coordinates.

Use of inheritance

Structure of the model:
Implement a model corresponding to the following UML class diagram, in particular the inheritance relationship. Various attributes have a random value.
Notice that all the get* are actions and R_choosePlot is a reflex of the **buyer** species.



Initialisation:

Create 1 buyer Maxi_buyer and 1 Rnd_buyer agents.
Create 10 agents of each subclass of plot.
Hint: initialise a global variable all_plots (type: list of plot), set with all the agents inheriting from plot, see the of_generic_species operator)

Dynamics:
The buyer reflex R_choosePlot calls the choosePlot action to choose one plot and write it in the console.
This choosePlot action is redefined in each subclass of buyer:

- In Maxi_buyer, the agent choose the plot that maximizes the value biomass * N.
- In Rnd_buyer, the agent creates a list of values with, for each plot, the value biomass * N. It then chooses a plot randomly with a probability depending to this value (hint: rnd_choice operator).

Think to define the various actions of plot.