

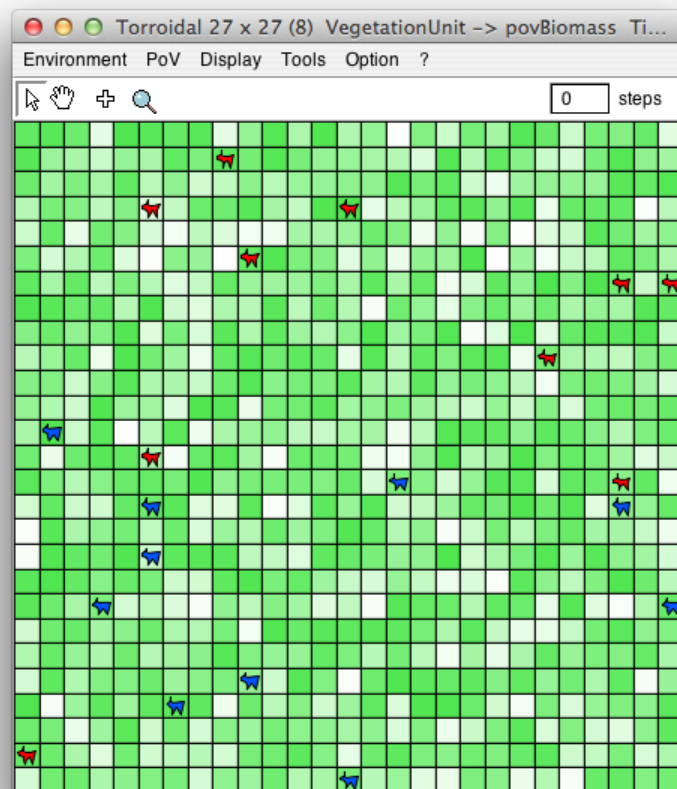
Building a Cormas model from scratch step by step: the ECEC model

CIRAD, GREEN Research Unit: http://www.cirad.fr/ur/green_en



**CORMAS (2012 release)
Common-pool Resources
and Multi-Agents Systems**

**Building a Cormas model from scratch step by step:
the ECEC model**



April 2012

Authors: Pierre Bommel, Christophe Le Page, Nicolas Bécu & François Bousquet

Table of contents

1. Model description	4
1.1. The Plants	4
1.2. The Foragers.....	5
1.3. Model formalism in UML.....	5
2. Adapting ECEC to Cormas framework.....	8
3. Implementing ECEC on Cormas	10
3.1. Opening Cormas.....	10
3.2. Creating a new Cormas model	10
3.3. Defining a spatial entity: the “VegetationUnit”	11
3.4. Designing the <i>VegetationUnit</i> behavior.....	13
3.5. Designing an agent foraging the resource.....	18
3.6. Setting the attributes of the foragers	19
3.7. Create two sub-classes of Forager.....	20
3.8. Coding the biological methods of Forager	22
3.9. Coding the “step” method of the forager agent.....	23
4. Designing methods to observe the entities	23
4.1. Assigning a green intensity according to the energy level of the plant	23
4.2. Assigning a color to distinguish foragers	26
5. Designing control methods for the scheduling of the simulation experiments.....	29
5.1. Create attribute agentsInitialNumber	29
5.2. Write a method to create the forager agents	30

Building a Cormas model from scratch step by step: the ECEC model

5.3. Write (in the “init” protocol) 3 different methods to be used as initial situations for simulation experiments.....	32
5.4. Write (in the “control” protocol) a step method to be executed at each time-step of simulation	33
6. Designing “probes” to record the variations of markers	34
6.1. Open the “probes” definition window from the main Cormas interface.....	34
6.2. Add probes based on attributes of the model (global level)	34
6.3. Write the code for the first one: a cumulative value over a collection.....	35
6.4. Define 2 other probes: the number of restrained and unrestrained foragers	36
7. Simulating the model.....	36
8. Saving, loading and versioning the model	39
8.1. Saving and versioning.....	39
8.2. Loading the model.....	40
9. Playing with the model	40
9.1. Changing the parameters’ values.....	41
9.2. Manipulating the agents on the grid.....	42
9.3. Executable activity diagram editor to modify the agents’ behavior.....	47
10. Analyzing the model	53
10.1. Simple stochastic analysis.....	53
10.2. Individual signature of parameters	55
10.3. Sorting the sensitivity of the model to parameters	58

The model we present here and that you will build, is inspired from a paper by Pepper and Smuts, "Evolution of Cooperation in an Ecological Context":

Pepper, J.W. and B.B. Smuts. 2000. "The evolution of cooperation in an ecological context: an agent-based model". Pp. 45-76 in T.A. Kohler and G.J. Gumerman, eds. *Dynamics of human and primate societies: agent-based modeling of social and spatial processes*. Oxford University Press, Oxford.

Pepper, J.W. and B.B. Smuts. 2002. "Assortment through Environmental Feedback". *American Naturalist*, 160: 205-213

The model consists of a two dimensional grid, wrapped in both axes to avoid edge effects. It contains two kinds of entities: plants and foragers. The main idea is the study of the survival of two populations of agents that depends on the spatial configuration.

This ECEC model can be found on the Cormas web site:

<http://cormas.cirad.fr/en/applica/ecec.htm>

1. Model description

The model contains two kinds of entities: plants and foragers.

1.1. The Plants

The **Plants** are created only once and have a fixed location. They do not move, die, or reproduce. A plant's only "behaviors" is to grow (and be eaten by foragers). The plants vary only in their biomass, which represents the amount of food energy available to foragers. At each time unit, this biomass level increases according to a logistic growth curve:

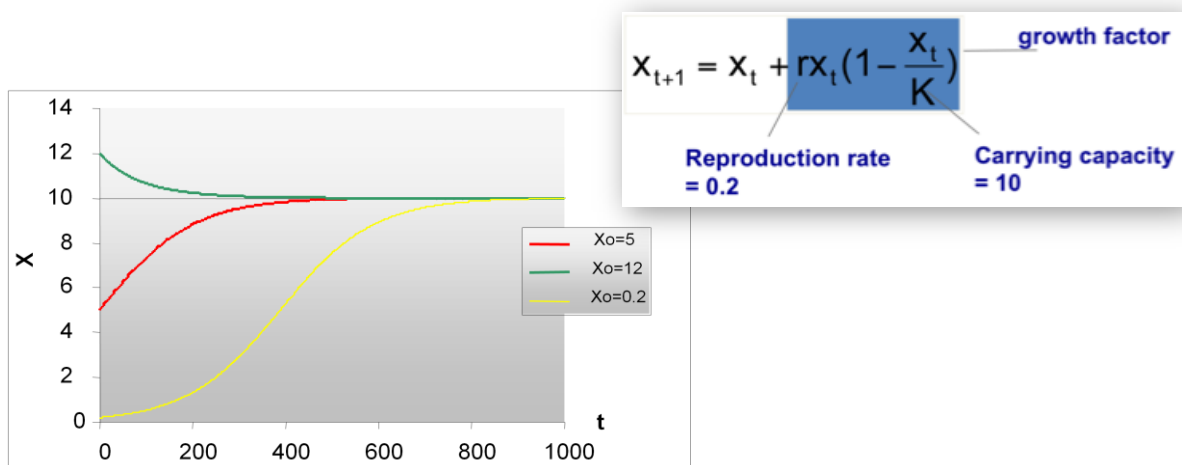


Figure 1: Logistic equation and its sigmoid curves

1.2. The Foragers

Each step, the Foragers burn energy according to their catabolic rate. This rate is the same for all foragers. It is fixed to 2 units of energy per time period.

A forager feeds on the plant in its current location if there is one. It increases its own energy level by reducing the same amount of the plant. Foragers are of two types that differs in their feeding behavior:

When “**Restrained**” foragers eat, they take only 50% of the plant’s energy.

In contrast, “**Unrestrained**” foragers eat 99% of the plant. This harvest rate is less than 100% so that plants can continue to grow after being fed on, rather than being permanently destroyed.

The Foragers do not change their feeding behavior type and their offspring keep the same heritable traits.

1.2.1. *Rules for Foragers’ Movements*

Foragers examine their current location and around. From those not occupied by another forager, they choose the one containing the plant with the highest energy.

If the chosen plant would yield enough food to meet their catabolic rate they move there. If not, they move instead to a randomly chosen adjacent free place (not occupied by another forager). This movement rule leads to the emigration of foragers from depleted patches, and simulates the behavior of individuals exploiting local food sources while they last, but migrating rather than starving in an inadequate food patch.

1.2.2. *Other Biological Functions of Foragers*

Foragers loose energy (catabolic rate, 2 points) regardless of whether or not they move.

If their energy level reaches zero, they die. But they do not have maximum life spans.

If a forager’s energy level reaches an upper fertility threshold (fixed to 100), it reproduces asexually, creating an offspring with the same heritable traits as itself (e.g., feeding strategy). At the same time the parent’s energy level is reduced by the offspring’s initial energy (50).

Newborn offspring occupy the nearest free place to their parent.

1.3. Model formalism in UML

1.3.1. *Structure of ECEC*

The following Class diagram presents the structure of the model.

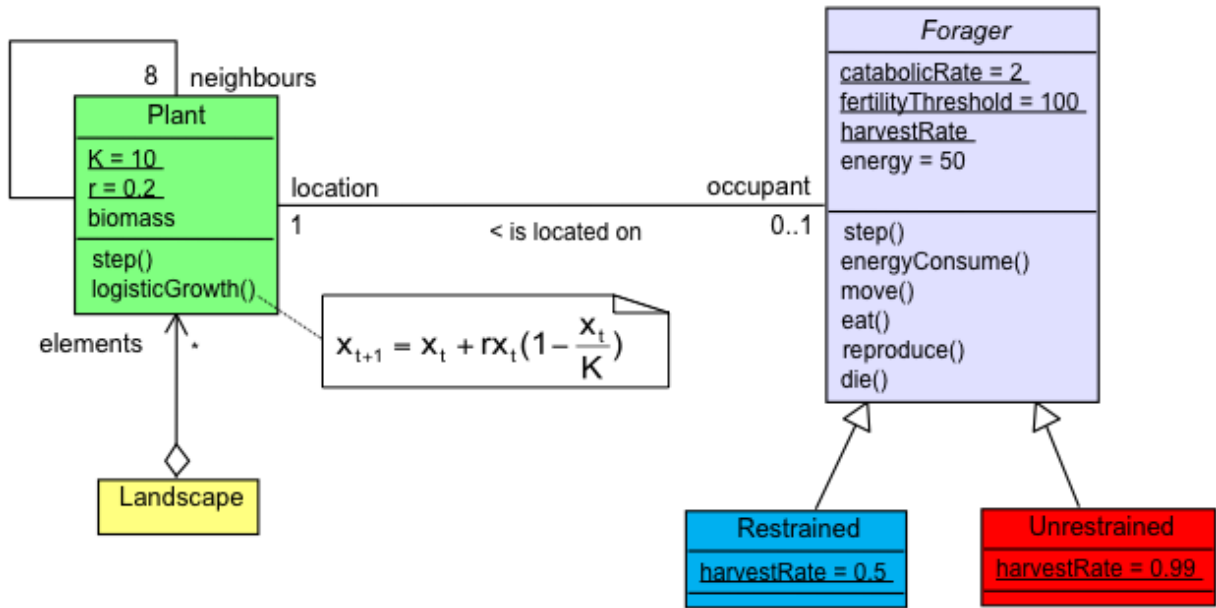


Figure 2: UML Class diagram of ECEC structure

The underlined attributes are called “Class variables”: their values are equal for all instances. For example, the catabolicRate class variable means that its value (2 units of energy) is identical for every foragers whatever their strategy (restrained or unrestrained).

1.3.2. Dynamics’ description of ECEC

The following Sequence Diagram presents the main time step of ECEC. This is a DTSS (*Discrete Time System Specification*, according to Zeigler et al. 2000¹ classification), meaning that, on the contrary of DEVS (*Discrete Event System Specification*), the evolution of the simulation is sliced in time steps.

As the model is purely theoretical, the step duration is not defined. In one step, all entities should evolve: the plants increase their biomass (according to Logistic equation), and the foragers perform their biological functions. In order not to give always preference to the same agents (the privilege to choose first the best plant), the list of the foragers is randomly mixed at each step.

¹ Zeigler, B.P., Praehofer, H. et Kim, T.G., 2000. (2nd ed.). Theory of modeling and simulation: integrating discrete event and continuous, Academic Press, New York

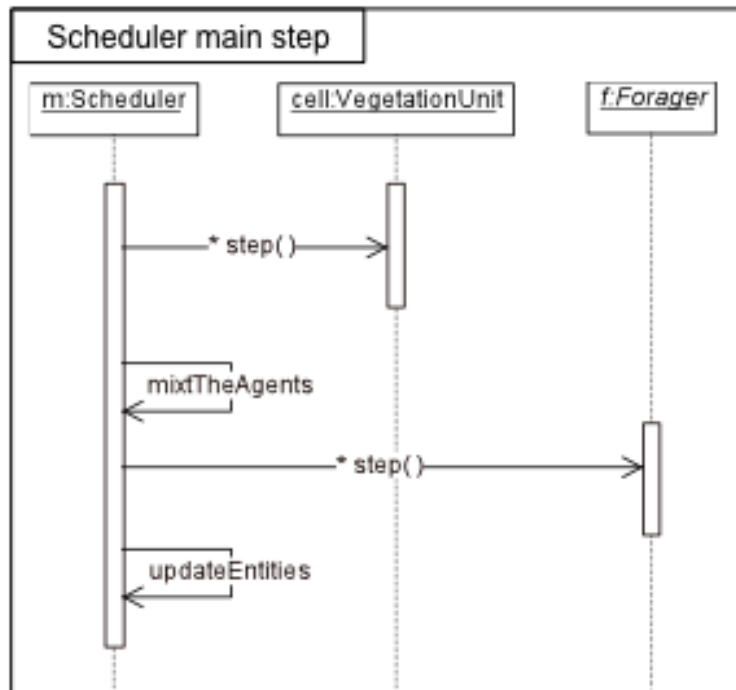


Figure 3: The main step Sequence diagram

At each step, a forager performs its activities:

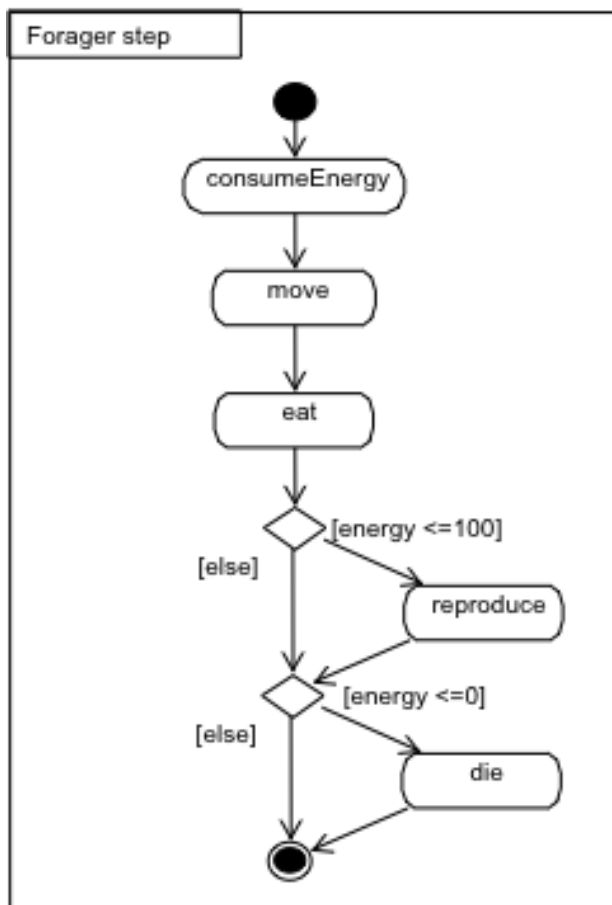


Figure 4: Activity diagram of a Forager

The follow Sequence diagram shows the “reproduce” behavior of a forager.

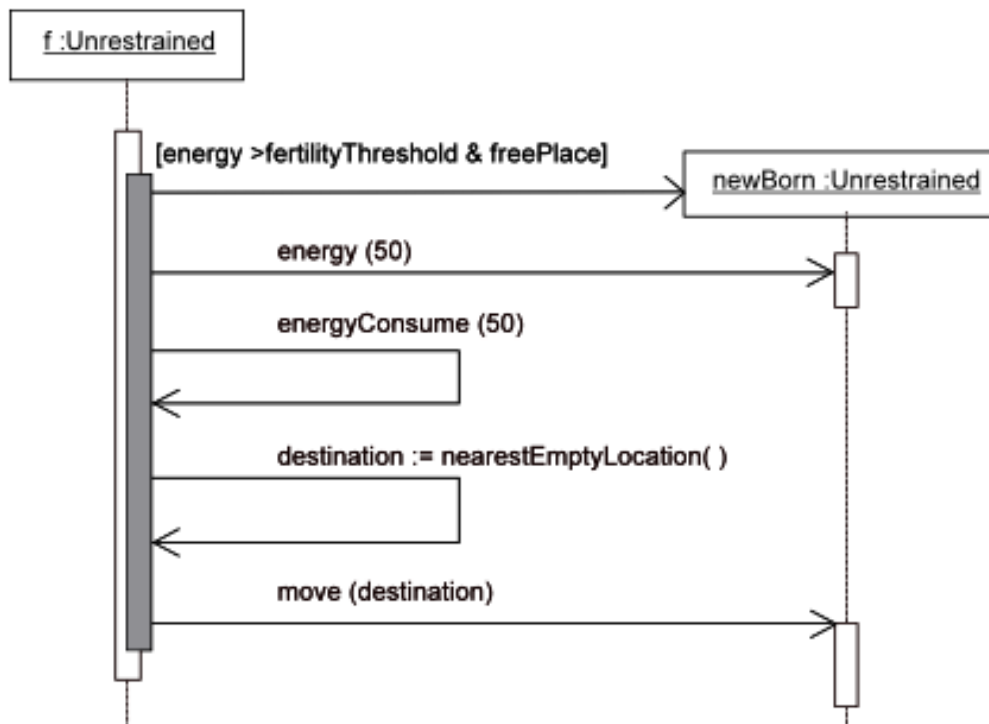


Figure 5: Sequence diagram of Reproduction process

2. Adapting ECEC to Cormas framework

The following diagram is an adaptation of the main class diagram (in design stage, see Figure 2) to fit the Cormas framework.

ECEC defines 3 kinds of entities: **Plants**, **RestrainedForagers** and **UnrestrainedForagers**. As the plants are spatially located and can't move, we aggregated the plant with the spatial unit in one entity. Thus, a **VegetationUnit** is a kind of *SpatialEntityCell* with additional attribute: “biomass”.

As the foragers are the located agents of ECEC, the **Forager** class must inherit from *AgentLocation* abilities, in order to enable the agents to move and to perceive their neighborhoods.

Building a Cormas model from scratch step by step: the ECEC model

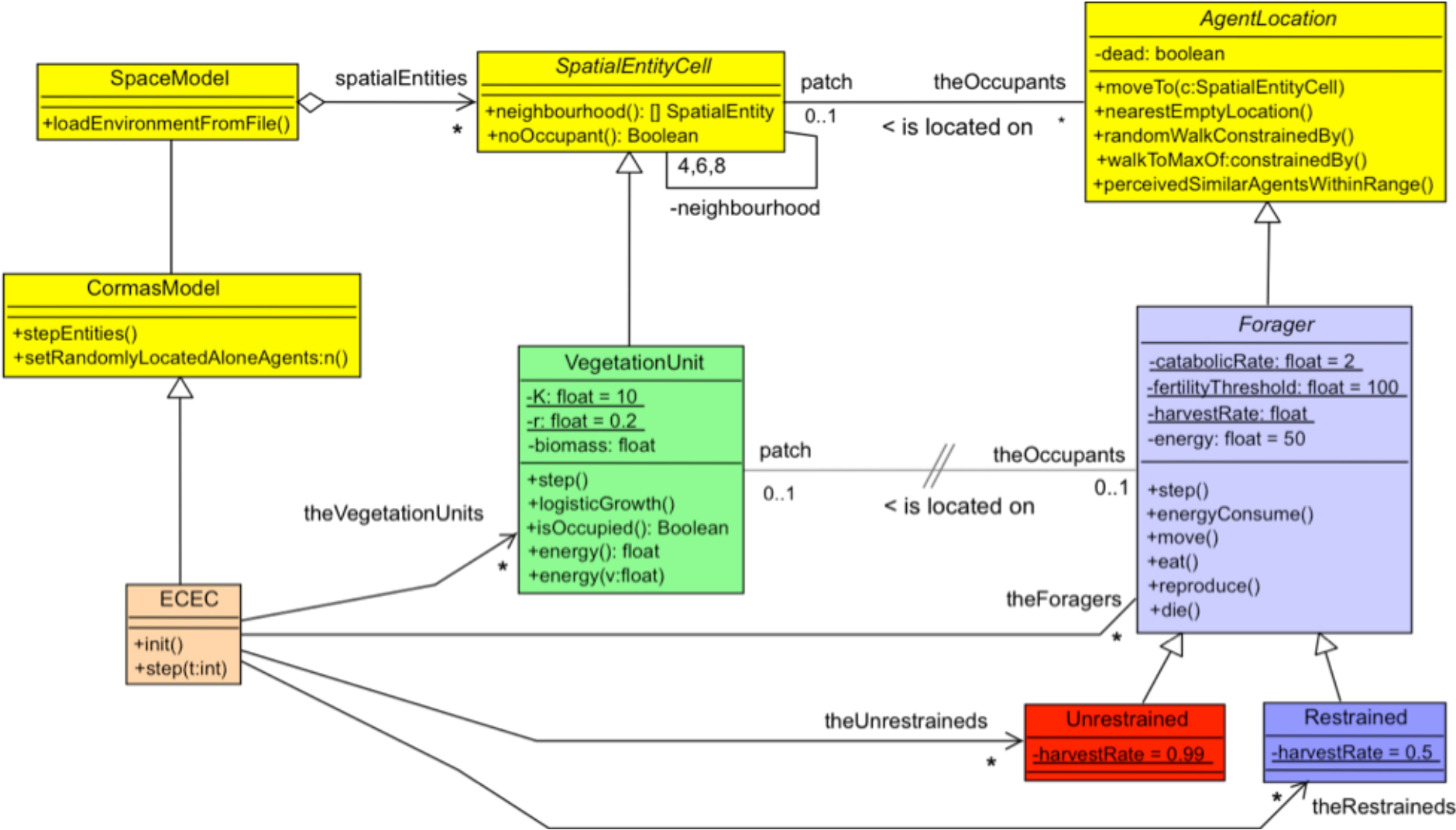


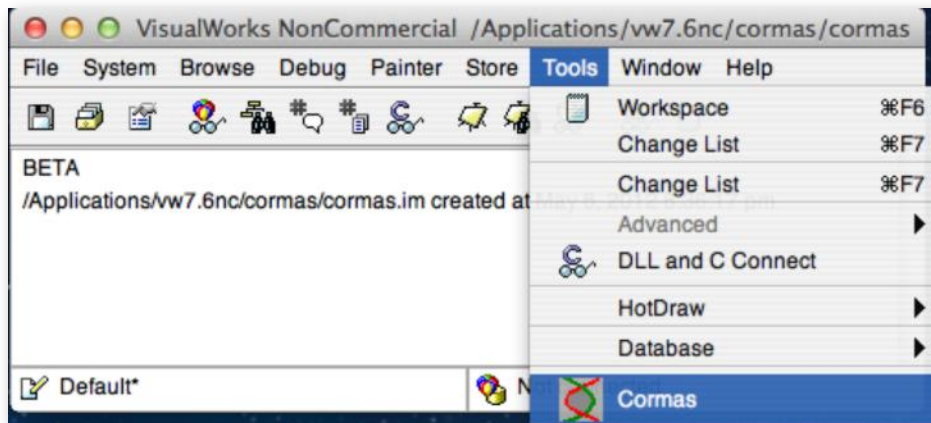
Figure 6: UML class diagram, adapted to Cormas (implementation stage)

3. Implementing ECEC on Cormas

As it is based on Smalltalk, an interpreted language, Cormas is a cross-platform software that can directly run on any platform without special preparation. Thus Cormas and your model should run on Microsoft Windows, Linux and Mac OS X. You can download the new release here: <http://cormas.cirad.fr/en/outil/download/newRelease.htm>

3.1. Opening Cormas

From the main interface of VisualWorks, select “Cormas” from the “Tools” menu:



Note that Cormas2012 does not propose the French version anymore.

While VisualWorks is minimized (but not closed), Cormas interface appears as follow:

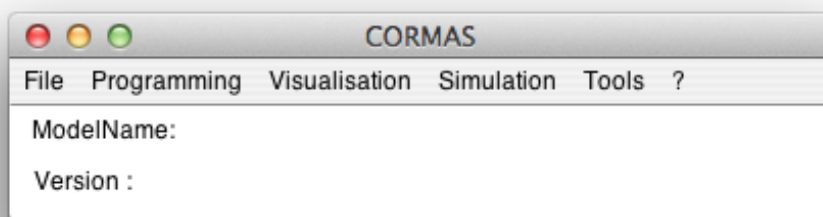


Figure 7: Cormas main interface

3.2. Creating a new Cormas model

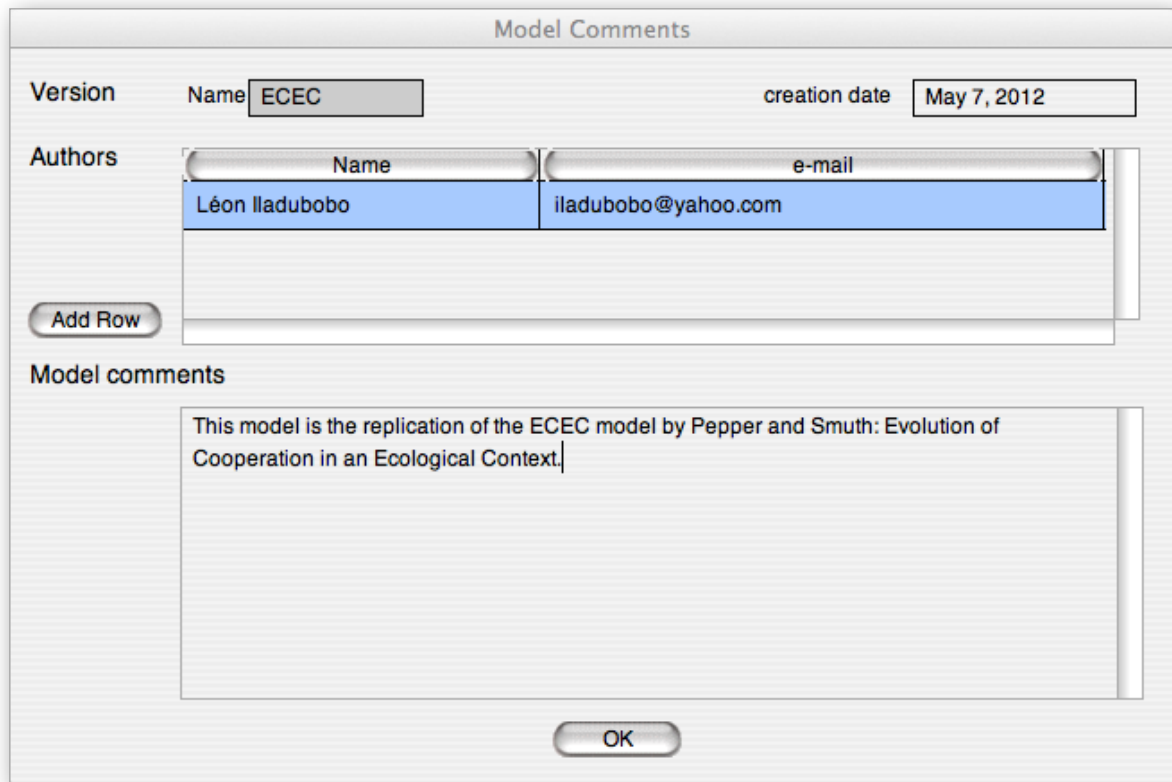
From Cormas menu, select *File* → *New*

Building a Cormas model from scratch step by step: the ECEC model



Thus, write the model' name: ECEC. Note that a model name must start with uppercase character.

After the click on “OK” button, a Comments interface pops-up. You can enter the authors' name and e-mails and write some comments on the purpose of this model, as following:



Then click the “OK” button. The next steps consist in programming the model.

3.3. Defining a spatial entity: the “VegetationUnit”

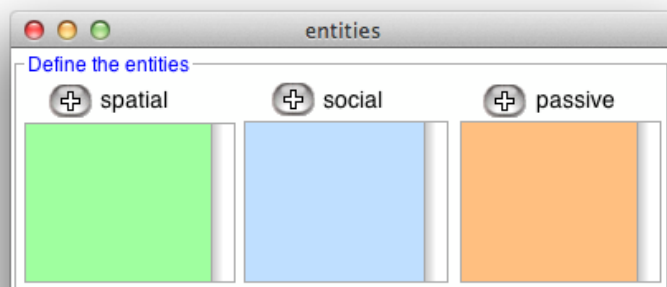
First of all, to define the entities, you must open the “*entities*” window: Select “*Program*” → “*The class for each entity*” on the main menu of the Cormas window:

Building a Cormas model from scratch step by step: the ECEC model



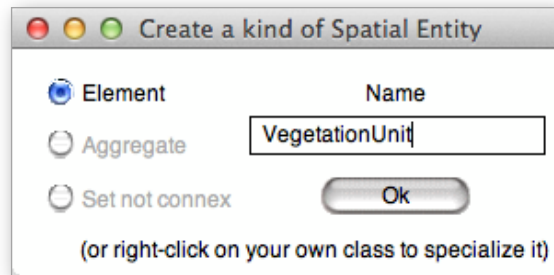
Figure 8: Cormas programming menu

Then the “entities” interface opens:



3 kinds of entities can be defined in Cormas: the spatial, social and passive (or other) entities.

To create a new kind of spatial entity (VegetationUnit), click on the [+] button near “spatial”. Then enter **VegetationUnit** in the input field and select “Element” as our class is a kind of SpatialEntityCell, the basic entity of the space:



Thus, the **VegetationUnit** class is created.

3.4. Designing the *VegetationUnit* behavior

3.4.1. *Setting the parameters of a logistic growth*

The logistic equation needs 2 parameters: the carrying capacity “**K**” and the growth rate “**r**”.

$$x_{t+1} = x_t + r x_t \left(1 - \frac{x_t}{K}\right)$$

As the values of these parameters are equals for each Plant instances, they are defined as class variables.

3.4.1.1. Create the K and r class variables

After selecting the *VegetationUnit* class, right-click on it to get a popup menu where you can Edit, Rename or Remove the selected class. In Edit sub-menu, select “Attributes”:

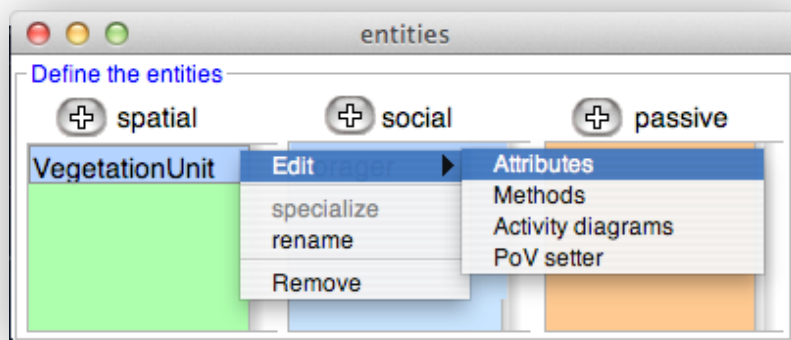
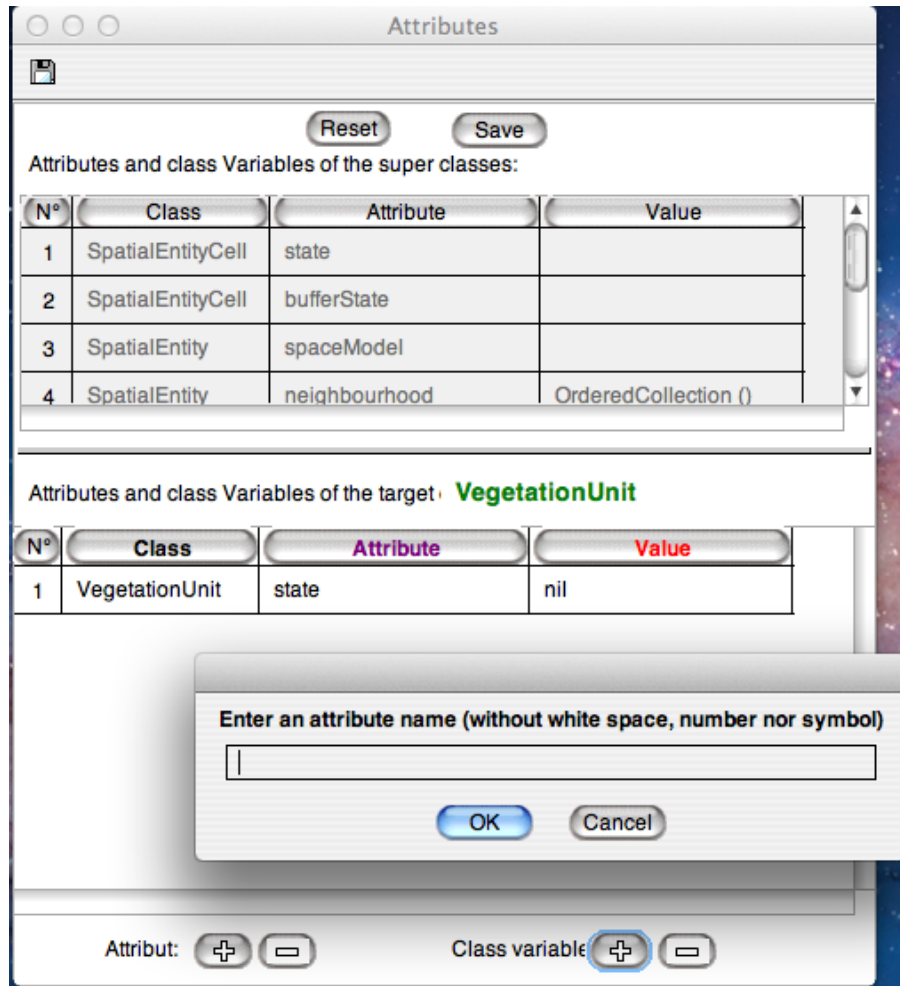


Figure 9: 'Entities' interface with contextual menu

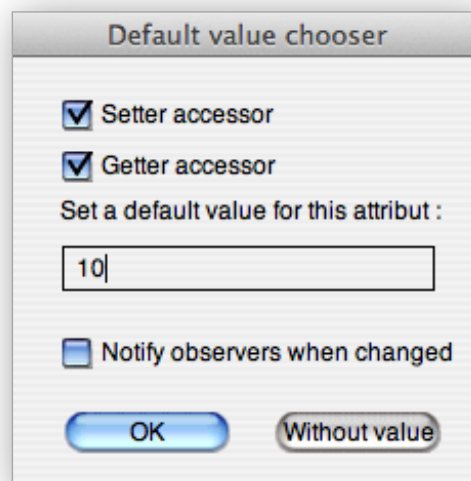
A new window opens to create and set attributes:

Building a Cormas model from scratch step by step: the ECEC model



Click on [+] button corresponding for Class variable, then enter **K** as new attribute name.

A default value chooser pops up. It is useful to set the default value of an attribute and to automatically create its accessors (public methods to read or change its value).



Building a Cormas model from scratch step by step: the ECEC model

As the carrying capacity of the plants is set to 10 (see fig. 1), enter 10 in the input field. You can deselect the “*Notify observers when changed*” button as K won’t change and is not observed during simulations.

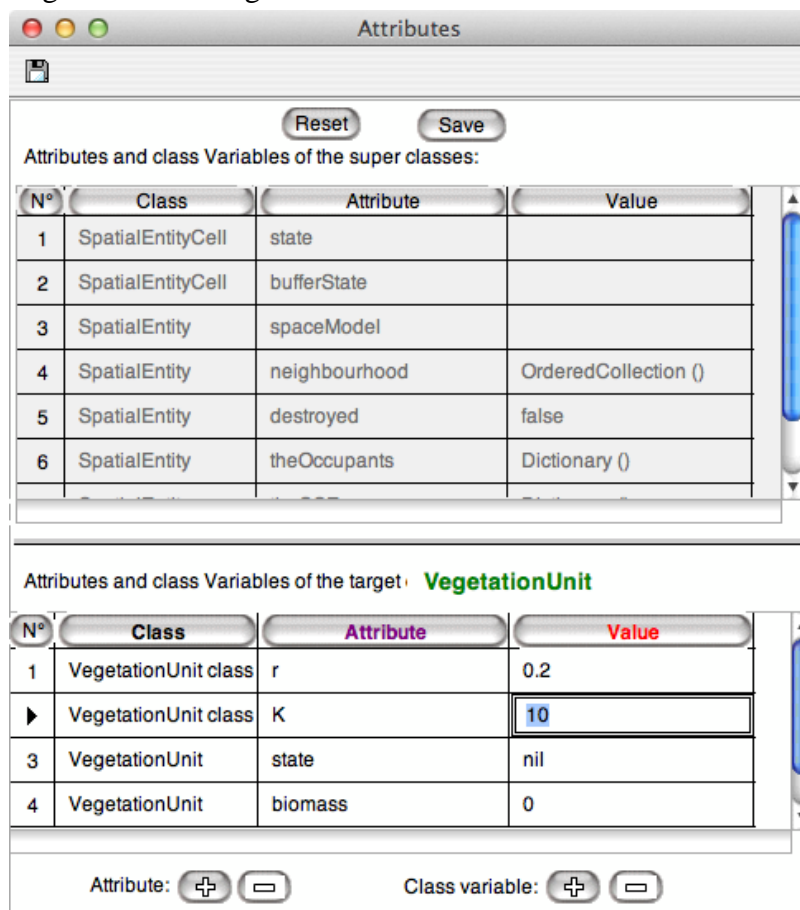
Repeat this process for **r** class variable, with 0.2 as value.

3.4.1.2. Create the *biomass* attribute

Similarly, click on [+] button corresponding for Attribute (bottom left), then enter **biomass** as new attribute name. The initial value of each instance of VegetationUnit cannot be defined here as it will depend on the way to initiate the landscape. Some cells may have very low biomass when other may have higher level of biomass. But, for sure, each one will have a number. So, let put 0 as default value.

As the biomass of the cells will be observed, let the “*Notify observers when changed*” button selected to show the vegetation changes during the simulations.

Thus, you should get the following window:



You can change the default value of the attribute from this interface. To change the value of K for example, select the target cell (10) and enter another number. Then click on “Save” button.

Building a Cormas model from scratch step by step: the ECEC model

You can also rename or remove attributes by right clicking on a row and by selecting “rename attribute” or “remove attribute” in the pop-up menu.

Now, close the Attribute window.

3.4.2. Write a logistic growth method in a new protocol

To code a method, you need to open a browser on the *VegetationUnit* class. For that, double click on *VegetationUnit* name of the ‘entities’ interface (or right click on it, *Edit* → *Methods*).

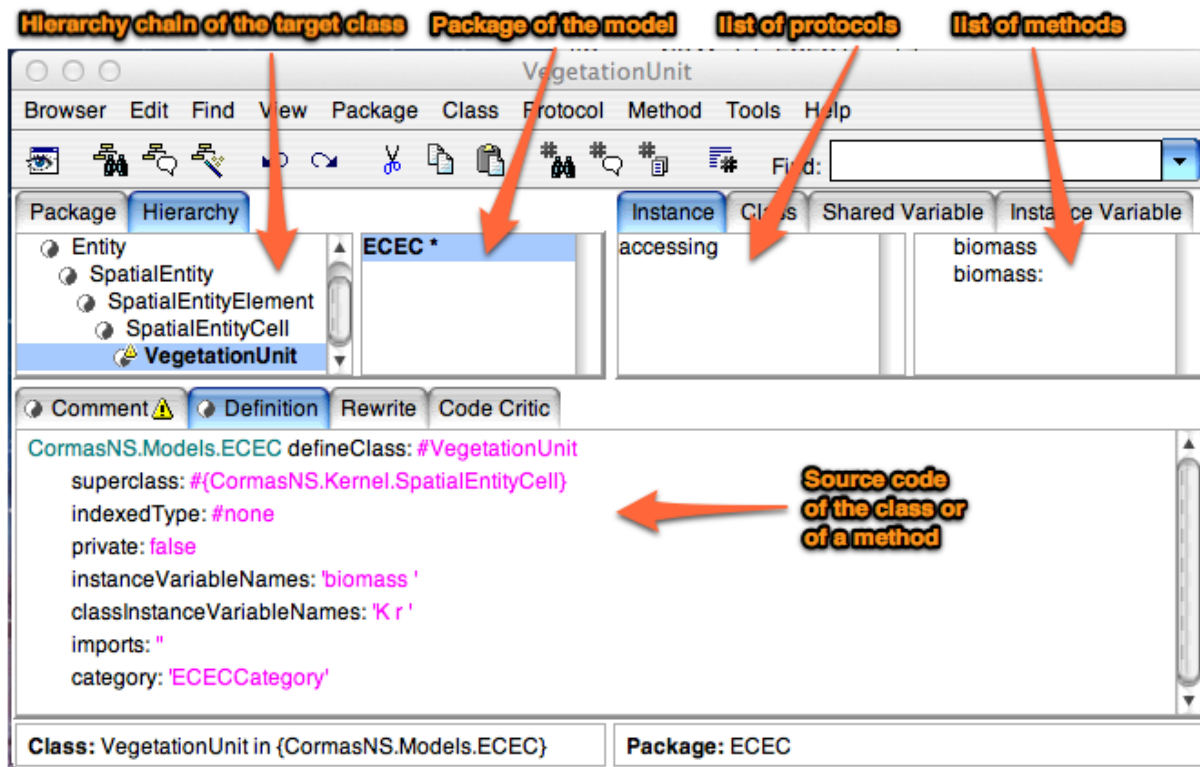
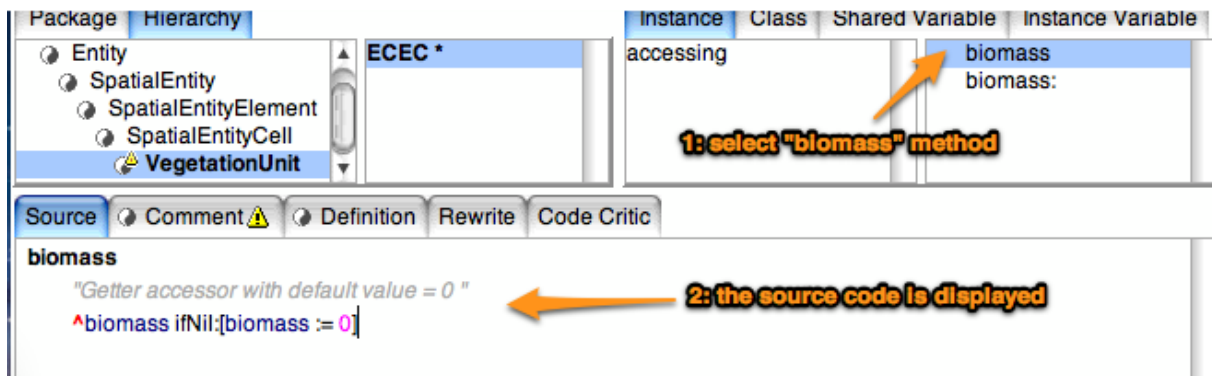


Figure 10: A browser of Smalltalk code

By default, the bottom panel displays the code that defines the class. By selecting the *biomass* method (into the methods panel), the bottom panel will display the code of this accessor method:

Building a Cormas model from scratch step by step: the ECEC model



#biomass and #biomass: are two methods to read and set the biomass value. They have been automatically generated by Cormas. Both methods are stored into the “*accessing*” protocol. But a protocol is just a way to organize the methods. To create a new protocol, right click on the "protocol" panel and select "new". Then write *growth* as protocol name.

To create a new method into the "growth" protocol, remove the text on the "source" panel (bottom) and write your own method on it:

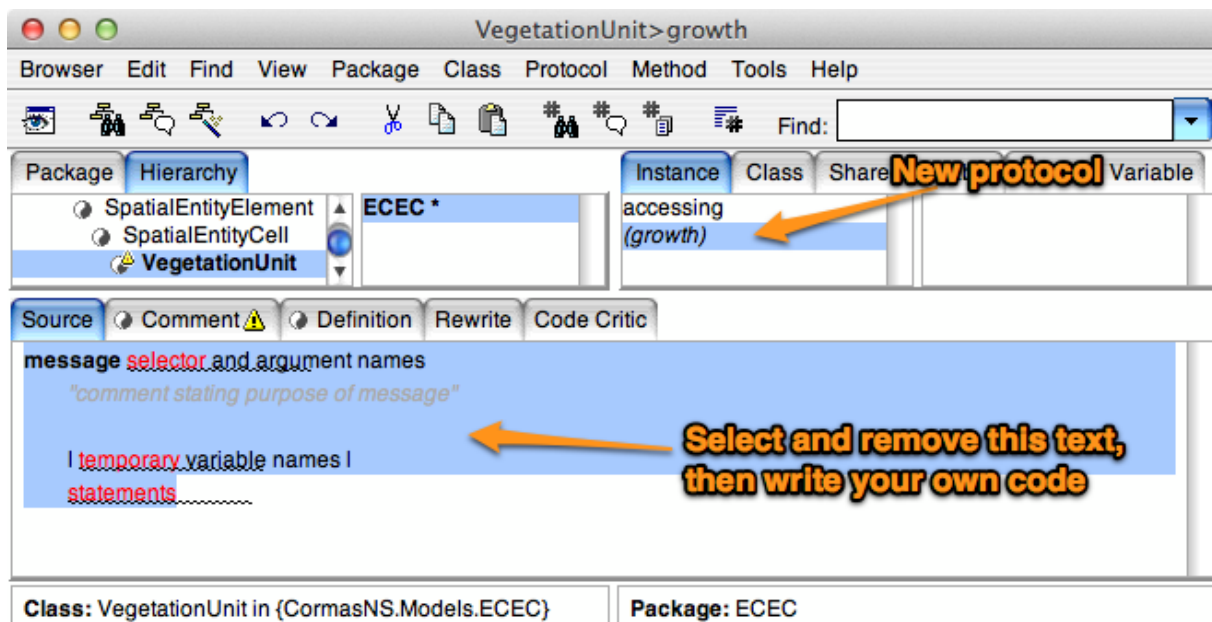


Figure 11: Creating a new protocol

Enter the following code:

`logisticGrowth`

```
self biomass: (Cormas logisticGrowth: self biomass r: self class r K: self class K)
```

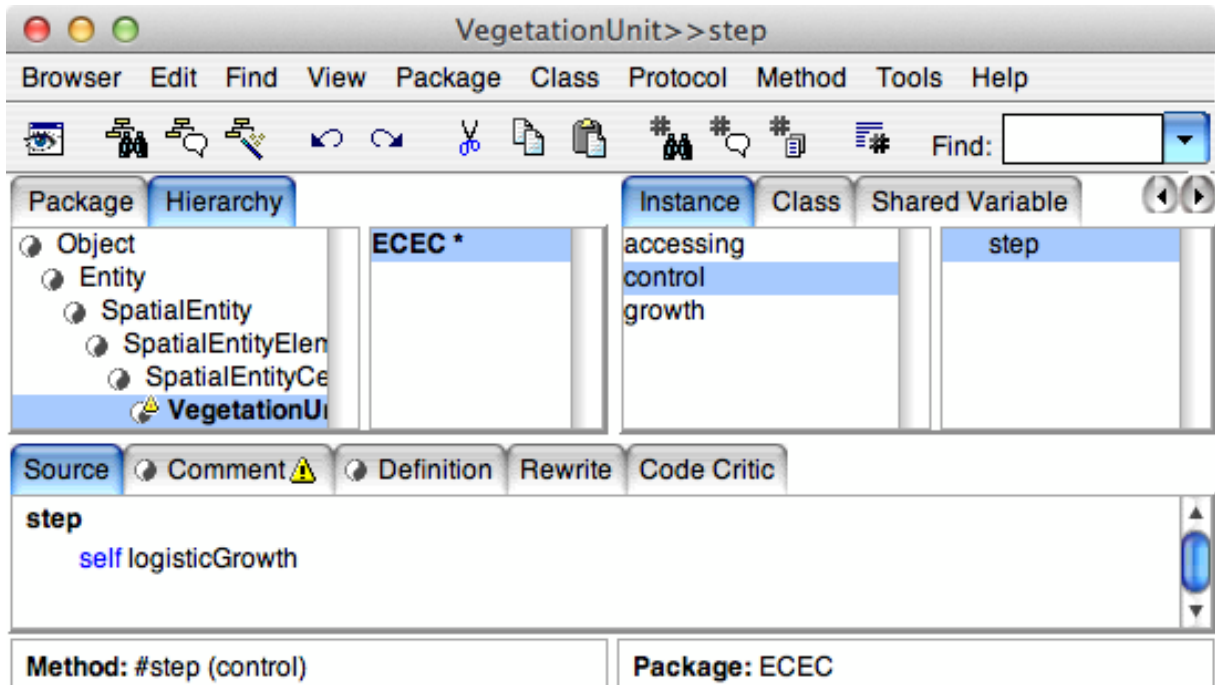
Then accept this code: right button → Acept or Ctrl S.

Building a Cormas model from scratch step by step: the ECEC model

The *logisticGrowth* method uses a static method from Cormas class (#logisticGrowth:r:K:).

3.4.3. Write the basic step method in the “control” protocol

In the same way as previously, create a new protocol called *control* and the *#step* method in it.



3.4.4. Write a random init method in the “init” protocol

In the same way as previously, create a new protocol called *init* and the *#initRandomBiomass* method in it.

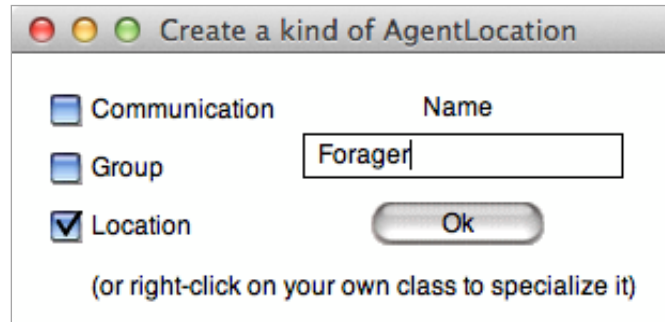
initRandomBiomass

```
"Set the initial value of biomass, between ]0 ; 1] ."
self biomass: Cormas random
```

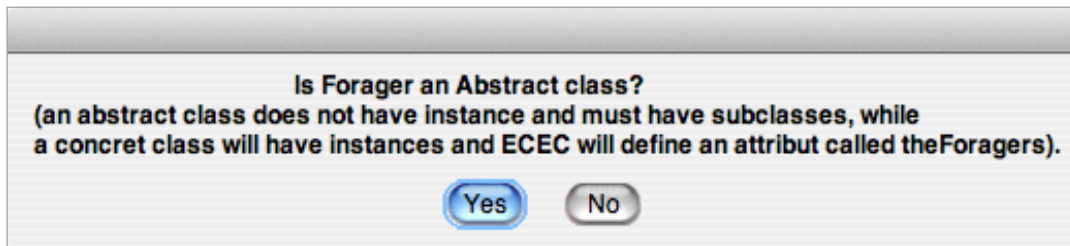
You can now close the browser of "VegetationUnit".

3.5. Designing an agent foraging the resource

Let us define the Forager agent as a kind of situated agent. On the second panel of ‘entities’ interface, click on [+] button, near “social”.



As the foragers are located, the **Forager** class must inherit from *AgentLocation*. Thus, select the “Location” button. When clicking “OK”, a message pops up, asking if Forager is abstract:



As we will redefine 2 sub-classes, click on “Yes”.

Then, as you can see in a new browser, Forager inherits from *AgentLocation*.

3.6. Setting the attributes of the foragers

The values of these parameters are equals for each agent. We can define them as class variables as describe in the class diagram (Figure 2).

3.6.1. Create the new class variables: *fertilityThreshold*, *catabolicRate* and *harvestRate*

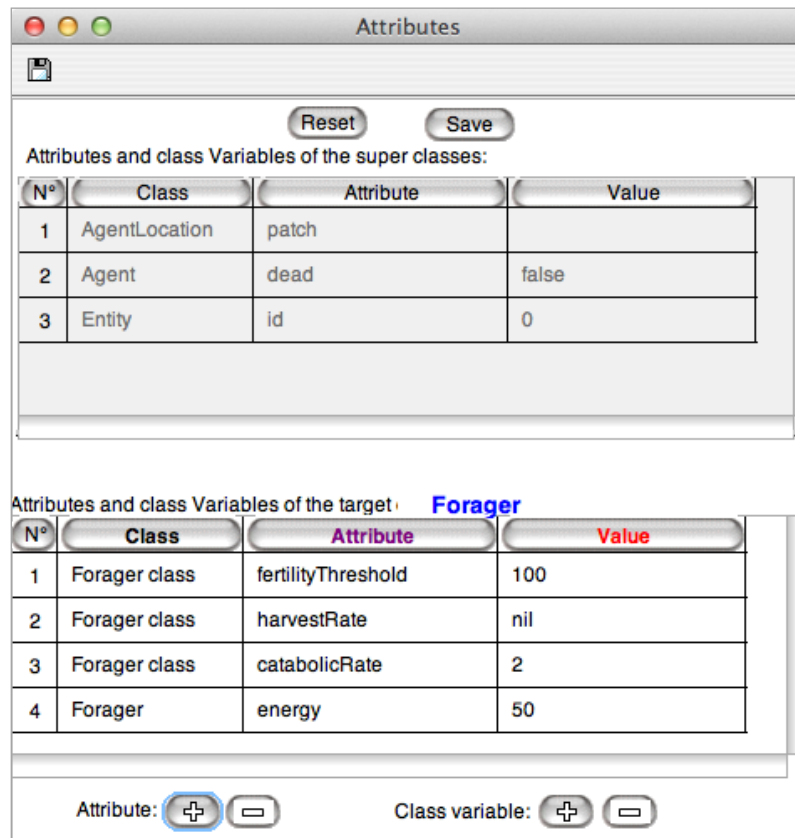
This is the same procedure as for *VegetationUnit* attributes: right click on Forager to display the contextual menu (cf. Figure 9). Select *Edit* → *Attributes*

By clicking on the [+] class variable button (bottom right), create the class variable *fertilityThreshold* with 100 as default value (without notification) and the *metabolicRate* with 2 (without notification). As *harvestRate* has no value assigned at this level, just add this class variable without value (click on “without value” button).

3.6.2. Add the “energy” attribute

Proceed like in section 3.4.1.2 (Create the *biomass* attribute): click on [+] button corresponding for Attribute (bottom left), then enter **energy** as new attribute name. The initial value of each instance is set to 50 (so, put 50 as default value).

As the energy of the foragers may be observed (display a color for energy level or display the value near each agent on the grid), select the “*Notify observers when changed*” button. Thus, you should get the following window:



Now that all parameters have been defined, close this window.

3.7. Create two sub-classes of Forager

As designed in the class diagram, Forager is specialized in two sub-classes: **Restrained** and **Unrestrained**. Because Forager is a class of *our* model, there is another procedure to create specialized classes: in “entities” interface, select Forager then right click on it and select “specialize”.

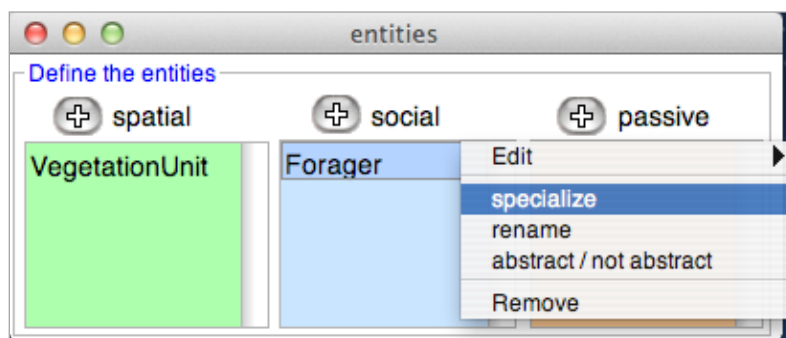
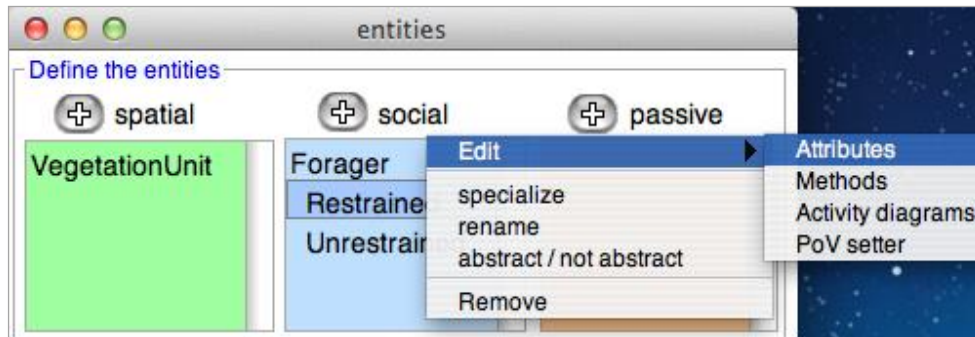


Figure 12: Create a subclass by specializing a super class of the model

Building a Cormas model from scratch step by step: the ECEC model

Then, enter the new class name: *Restrained*. Because there will be instances of restrained foragers, this class is not abstract: click “No” to the question “is *Restrained* an Abstract class?”. Repeat this procedure for *Unrestrained*.

Now, we must define the value of *harvestRate* for each subclass of *Forager*. To do that, reopen the attribute editor for each one. Select a subclass (*Restrained* for instance), edit the contextual menu with *right click* → *Edit* → *Attributes*:



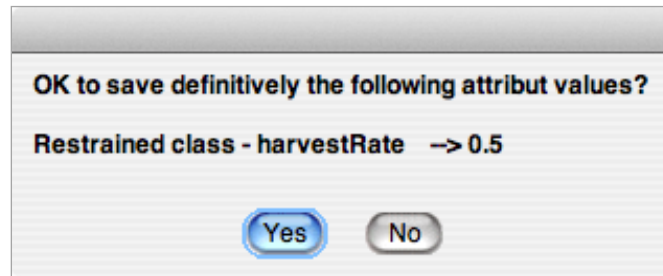
Then, at the row corresponding to *harvestRate*, change the **nil** value for **0.5**.

N°	Class	Attribute	Value
1	Forager class	fertilityThreshold	100
2	Forager class	harvestRate	
3	Forager class	catabolicRate	2
4	Forager	energy	50
5	AgentLocation	patch	
6	Agent	dead	false

Attributes and class Variables of the target: Restrained			
N°	Class	Attribute	Value
1	Restrained class	fertilityThreshold	100
▶	Restrained class	harvestRate	0.5
3	Restrained class	catabolicRate	2
4	Restrained	energy	50

Attribute: Class variable:

Then, click on “Save” button. By clicking on “Yes” button of the following question...



...Cormas will redefine the getter accessor for *harvestRate* with 0.5 as default value.

Close the Attributes window and repeat this procedure for **Unrestrained** by setting the default value to **0.99**.

3.8. Coding the biological methods of Forager

Similarly to the Logistic growth of *VegetationUnit*, we have to code the biological methods of the foragers. As described in the main class diagram, these methods are equals for both subtypes of foragers. Thus, they have to be defined at *Forager* level.

Edit the *Forager* class and create “biology” protocol (see chapter 3.4.2, Write a logistic growth method in a new protocol, page 16). Then write the following methods.

consumeEnergy

"Loose energy according to catabolic rate (-2 by time step)"

```
self energy: self energy - self class catabolicRate
```

move

"The Forager examines its current location and around. From those not occupied, he chooses the one containing the plant with the highest energy. If the chosen plant would yield enough food to meet their catabolic rate (2 units), he moves there. If not, he moves instead to a randomly chosen adjacent free place (not occupied by another forager)"

```
l goodCells l
```

```
goodCells := self patch neighbourhoodAndSelf select:
```

```
[:cell l cell biomass > self class catabolicRate and: [cell noOccupant]].
```

```
goodCells isEmpty
```

```
ifTrue: [self randomWalkConstrainedBy: [:c l c noOccupant]]
```

```
ifFalse: [self moveTo: (goodCells asSortedCollection: [:c1 :c2 l c1 biomass > c2 biomass]) first]
```

eat

"eat a quantity of the biomass of the cell"

```
l qty l
```

```
qty := self patch biomass * self class harvestRate.
```

```
self energy: self energy + qty.
```

```
self patch biomass: self patch biomass - qty
```

reproduce

"forager reproduces asexually, creating an offspring with the same heritable traits as itself (e.g., feeding strategy). At the same time the parent's energy is reduced by the offspring's initial energy (50). Newborn offspring occupy the nearest free place to its parent. "

```
I newForager freePlace I
freePlace := self nearestEmptyLocationWithinRadius: 1.
freePlace ifNil: [^nil].
newForager := self newEntity: self class.
self energy: self energy - newForager energy.
newForager moveTo: freePlace
```

die

```
"set dead attribute to true"
self dead: true
```

3.9. Coding the “step” method of the forager agent

step

```
self consumeEnergy.
self move.
self eat.
self energy >= self class fertilityThreshold ifTrue: [self reproduce].
self energy <= 0 ifTrue: [self die]
```

As #step is already defined into Agent class, the new #step is automatically moved into the “* control” protocol.

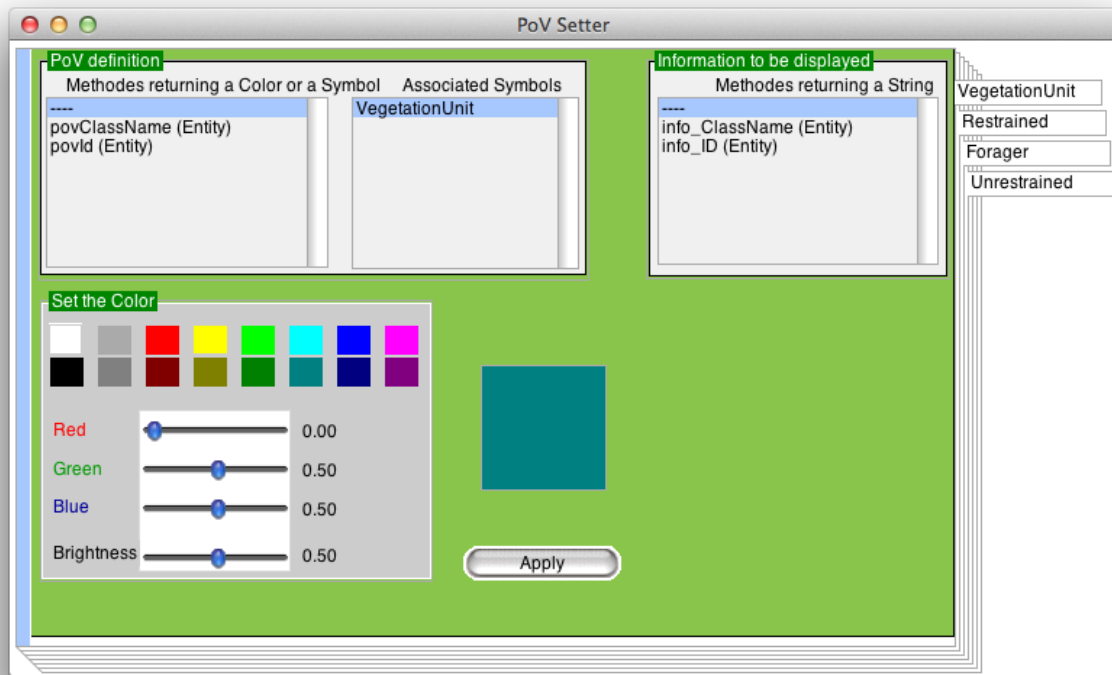
Due to the new diagram editor, there are two ways to design the main behavior: by coding or by creating an activity diagram. The diagram editor will be presented later (chap. 9.3, p. 47).

4. Designing methods to observe the entities

4.1. Assigning a green intensity according to the energy level of the plant

4.1.1. Add a “point of view” method named “povBiomass”

A “Point of View” setter (PoV Setter) can be opened by two ways: from the Cormas menu, select *Programming* → *the observer* → *Space_and_entities*, or from the Entities interface, right click on a *class* → *Edit* → *PoV_setter*.



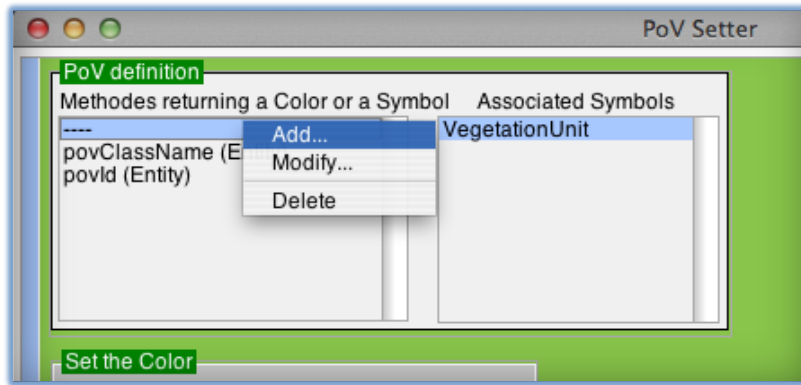
By default, #VegetationUnit (a symbol associated to the class name) has been created by Cormas. You can choose a color for this name for future use, even if we won't use it for the ECEC model. For that, select the #VegetationUnit symbol, click on the desired color and click on "Apply" button.

4.1.2. Write the code of the "povBiomass" method

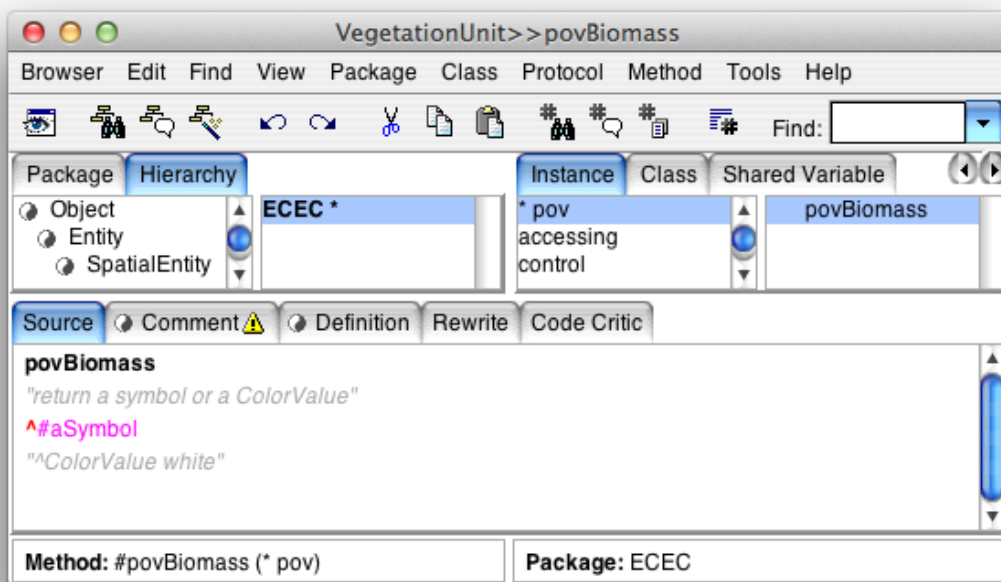
A "pov" method has to return a symbol, which needs to be associated to corresponding color via the palette tool. Here, as we want a gradient color related to the biomass of a cell, we won't use the classical way of writing "pov" methods, but a "pov" method that returns directly a color value. The Entity class provides a generic method that relates the brightness of a base color to the value of a quantitative attribute: #povAttribute:min:max:color:.

But first of all, let's create a new PoV method, that, when selected, will display a gradient of green color according to the biomass of each cell. On "PoV definition", right-click and write "povBiomass":

Building a Cormas model from scratch step by step: the ECEC model



After clicking on OK, a new browser pops up:



povBiomass

"return a Green Color gradient, according to #biomass"

```
^self  
povAttribute: #biomass  
min: 0  
max: self class K  
color: ColorValue green
```

The first argument helps to determine the *attribute* of the cell for which the color gradient is set. Second and third arguments are to set the *limits* of the gradient. We have chosen 0 for the minimum (a white color) and K for the darkest green. The last argument determines the *color*.

4.1.3. *Adjust the write-access method of all the attributes involved in “pov” methods*

The only attribute involved in a “pov” method of the **VegetationUnit** entity is “biomass”. To ensure that the visualization will be updated each time a new value of biomass will be assigned to a plant, the “self changed” instruction must be present in the write-access method:

Because we have chosen to “*Notify observers when changed*”, when the biomass attribute was created, this instruction has already been added by Cormas:

```
biomass: anObject  
"Setter accessor of attribute biomass "  
biomass = anObject ifTrue:[^nil]. "for optimization"  
biomass := anObject.  
self changed
```

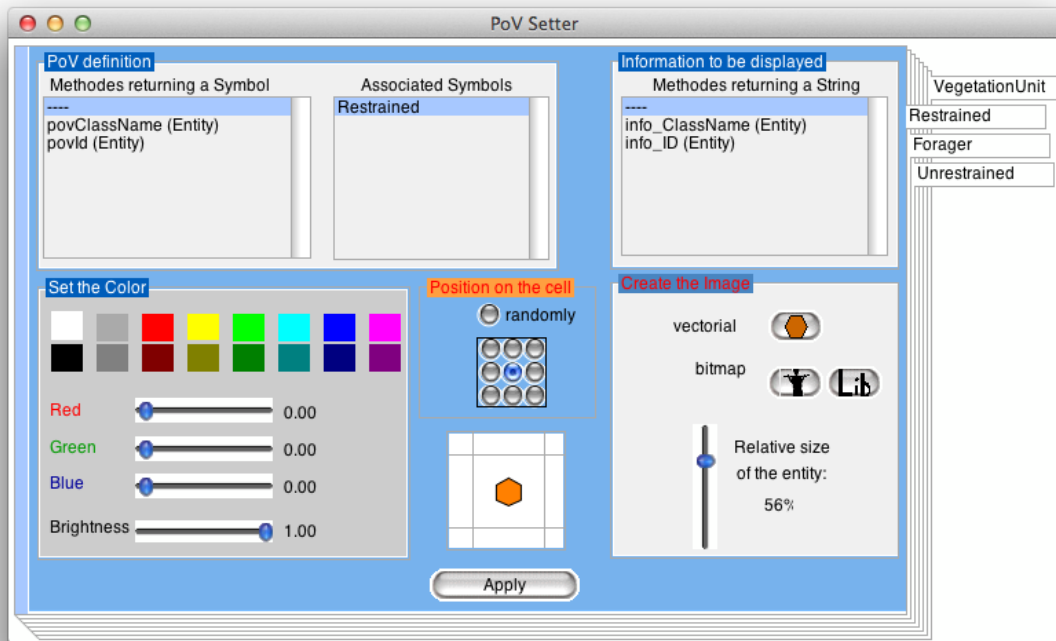
To automatically add (or remove) this instruction for an attribute, open the attribute setter (Figure 9, p. 13, right-click on a *class* → *Edit* → *Attributes*), then select the row of the target attribute and select “*Notify observers when changed*” while right-clicking on it.

4.2. Assigning a color to distinguish foragers

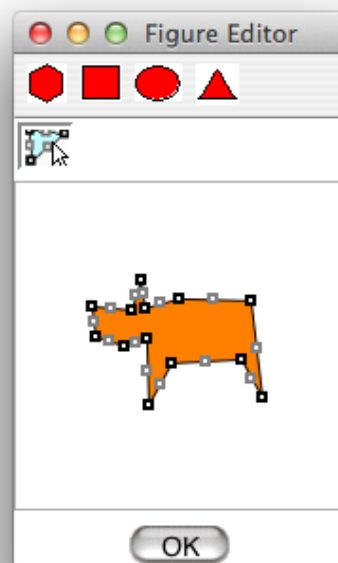
4.2.1. *Associate a “pov” each type of Forager*

Reopen a PoV Setter (or select it) and select the “Restrained” item at the right of the window.

Building a Cormas model from scratch step by step: the ECEC model



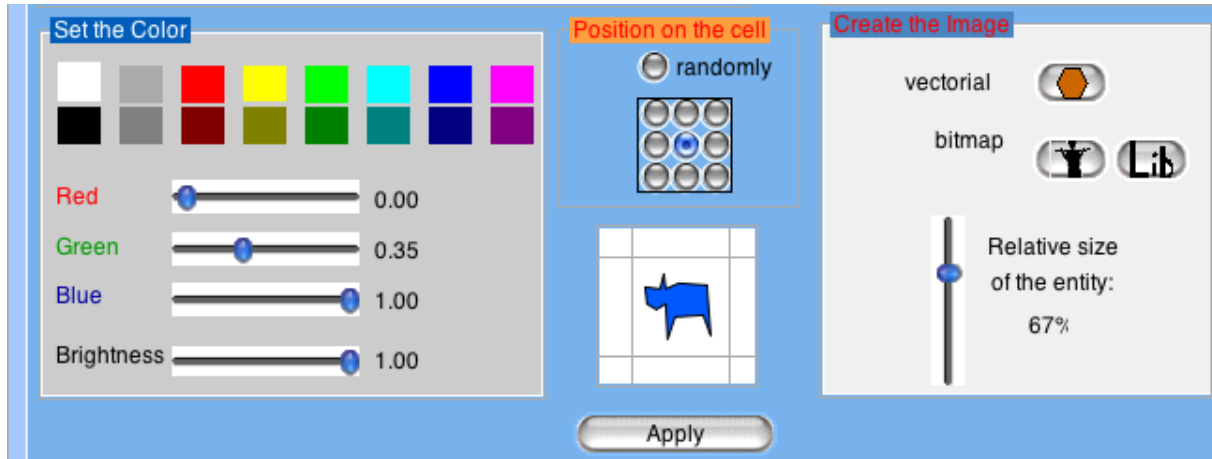
As the `#povClassName` method (already defined for Entity) returns the name of the class, thus `#Restrained` symbol is already created, and the default image associated to this symbol is a hexagon. You can choose the color of this hexagon by clicking on the color setter, then “Apply”. You can also design a shape for the Restrained foragers. For that, click on the “Vectorial” button into the “Create the image” box, then select the initial shape (one of the 4 buttons on the top) and modify it with your mouse:





Building a Cormas model from scratch step by step: the ECEC model

When it is done, click on “OK”. You can then

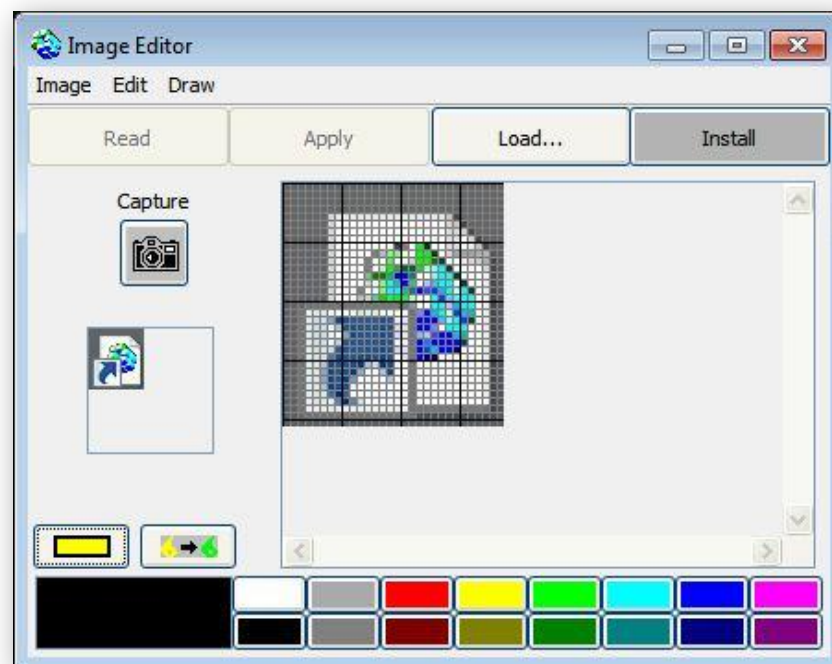
- resize the image,
- change the color
- define its position on the cell.



After a click on “Apply” (don’t forget !), repeat the procedure for the Unrestrained forager.

You may also use bitmap images for the agents, using the bitmap library  or the Image editor .

By clicking on the “Image Editor” button, a new window pops up:



After having clicked on the “Capture” button, you can click and drag your mouse on an image displayed by your screen (note that you must move the mouse from the top-left to the bottom-right position). When it’s done, click on the “Install” button.

4.2.2. Add the symbols that may be returned by the “pov” methods

In the “Symbols” window, you need to add all the possible symbols that may be returned by any “pov” methods of the selected entity. To each added symbol, associate a color and a shape. Use the palette or the Red, Green and Blue sliders to choose a color.

For the **Unrestrained** forager, you may decide to have the same shape than for the **Restrained**. For that, right-click on the “Associated symbols” box and select “Add – same shape as” → *Restrained* → *Restrained*, then type *Unrestrained* for the new symbol. You can then change the color, resize the figure or even modify it. Do not forget to apply after your modifications.

5. Designing control methods for the scheduling of the simulation experiments

Now it is time to design the model scheduler that is in charge of organizing the simulations, that is to say,

to initialize a simulation by creating the virtual world: to instantiate the agents and the environment and to set the links between the entities,

to schedule the entities by sending a *step* to each one. (Cormas is oriented towards *step-by-step* simulations, but it is possible to program *discrete events* simulations). For these reasons, the scheduler may be seen as an orchestra conductor.



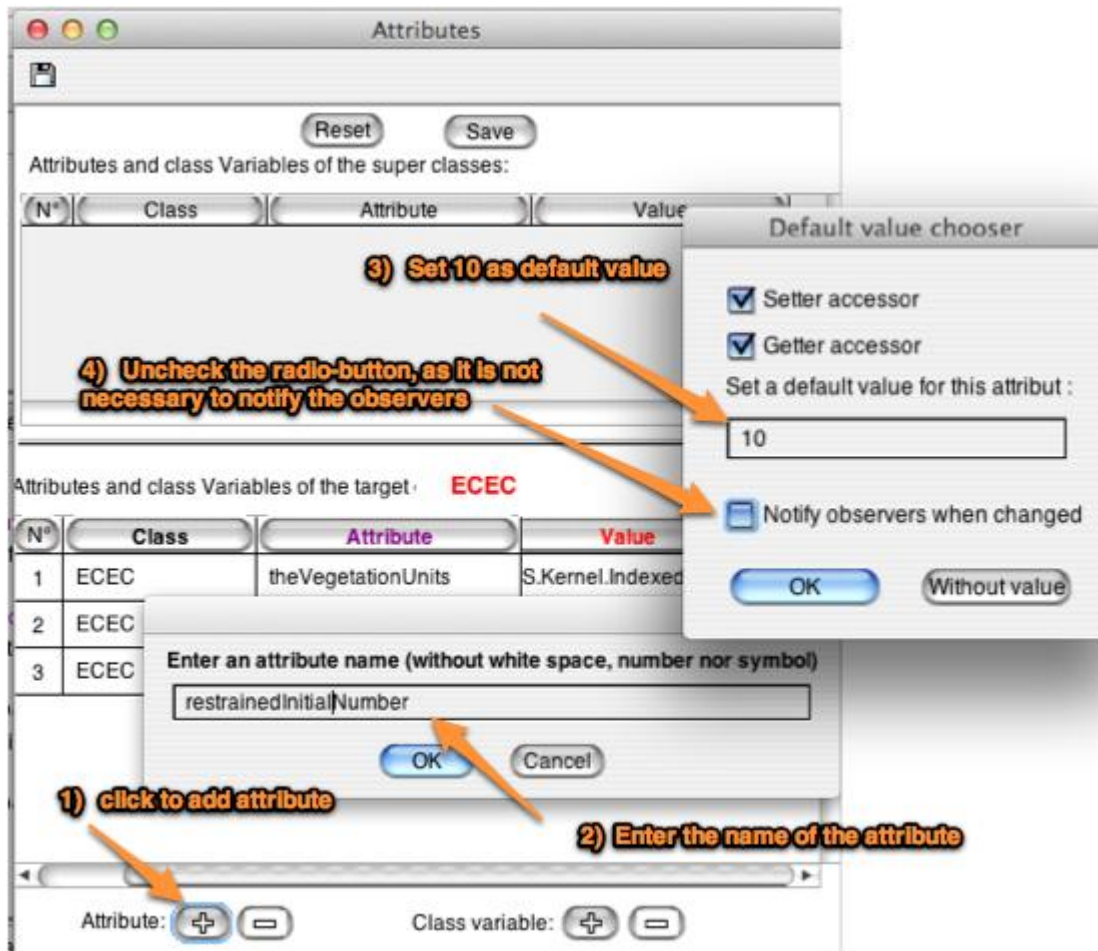
In Cormas, the scheduler class is the main class of your model, i.e. the ECEC class.

5.1. Create attribute *agentsInitialNumber*

Create the *agentsInitialNumber* attributes that allow the user to choose the initial number of each type of Foragers.

Building a Cormas model from scratch step by step: the ECEC model

Open the Attributes editor: Cormas *main menu* → *Programming* → *The simulation organization* → *The initial instantiation* → *Edit attributes*. Then click on the “+” button at the right bottom (for ‘Attribute’), enter *restrainedInitialNumber* as new attribute name and set the initial value equals to 10:



Repeat this procedure for *unrestrainedInitialNumber*, then close the attributes editor.

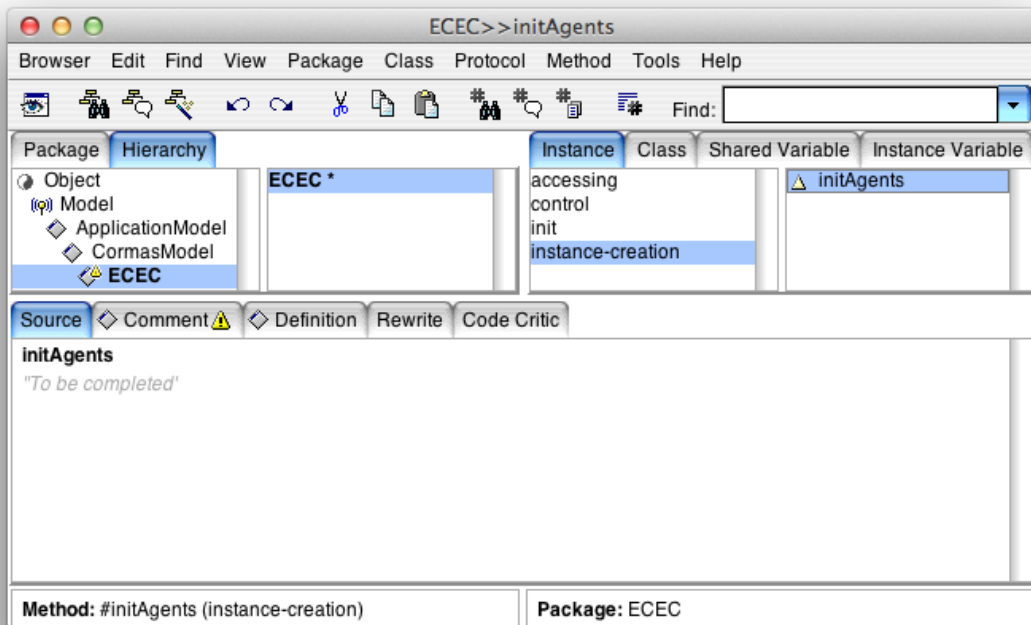
5.2. Write a method to create the forager agents

Now, let's write a method to create 10 Restrained and 10 Unrestrained foragers. As this new method is not an initialization method (callable from the simulation interface) but just a part of the simulation (we also need to initialize the space and the plants), we need to create a new protocol (“entities creation”) different.

Open a code browser on ECEC: Cormas *main menu* → *Programming* → *The simulation organization* → *The initial instantiation* → *Edit initialization*.

By default, the browser opens on the *init* protocol and targets the *#init* method of ECEC (4 protocols have been automatically generated by Cormas: accessing, control, init and instance-creation). Click on the instance-creation protocol, then on *#initAgents* method:

Building a Cormas model from scratch step by step: the ECEC model



Re-define the #initAgents methods: remove the comments *"To be completed"* then write:

```
self createN: self restrainedInitialNumber randomlyLocatedAloneEntities: Restrained.
```

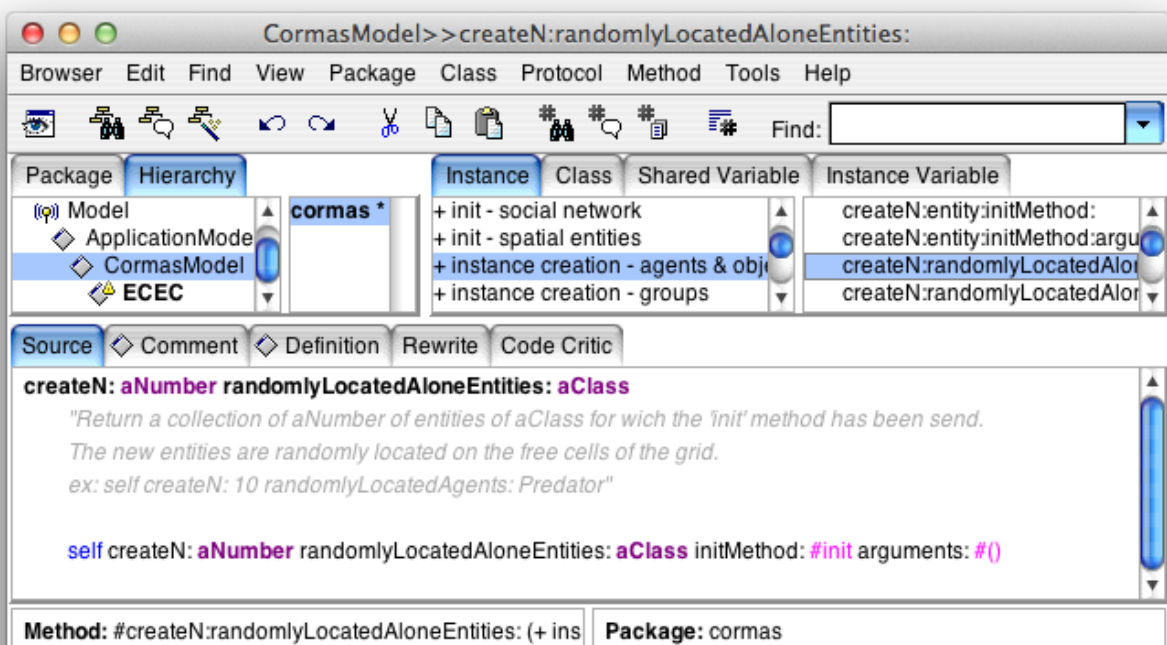
And a similar line instruction for the Unrestrained. Then Accept (Ctrl S). Thus, the new #initAgents method is saved:

initAgents

"initialize 10 foragers of each type and move them randomly on a free cell"

```
self createN: self restrainedInitialNumber randomlyLocatedAloneEntities: Restrained.
self createN: self unrestrainedInitialNumber randomlyLocatedAloneEntities: Unrestrained
```

Explanation: this method instantiates 10 Restrained and 10 Unrestrained foragers. It uses the #createN:randomlyLocatedAloneEntities: method that is already defined into the ECEC super class: CormasModel (you can click on this class in the first panel of the browser and navigate in it ; this method is into the + instance creation - agents & objects protocol).



This method expects 2 arguments: a number (for the number of new entities to be created) and a class (in our case, Restrained or Unrestrained). But the comments should be explicit:

"Return a collection of aNumber of entities of aClass for wich the 'init' method has been send. The new entities are randomly located on the free cells of the grid.
ex: self createN: 10 randomlyLocatedAgents: Predator"

5.3. Write (in the "init" protocol) 3 different methods to be used as initial situations for simulation experiments

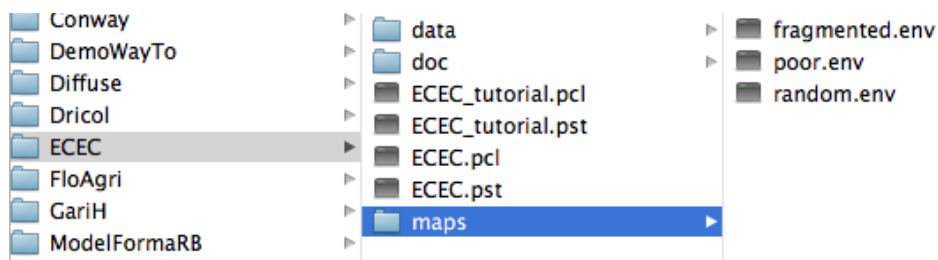
Here, we will write the methods that will be available for the user to initialize a simulation. The first one (#noForagers) will be useful to run a simulation without forager, just to show an increase of the vegetation. The second one (#homogeneousEnv) will be the basic initial state where the agents (10 and 10) are distributed on an homogeneous environment (the initial biomass of the plant is set randomly). For the last initial state, the environment is fragmented as patches of vegetation. You can load predefined grids or create a specific one.

5.3.1. Load predefined environments

To load these 3 environments, you need to download the maps.zip file on Cormas web site:

<http://cormas.cirad.fr/logiciel/maps.zip>

Unzip maps.zip into Models/ECEC/ in order to get the following hierarchy:
cormas/Models/ECEC/maps :



Then write the 3 initialization' methods. As we want them to be displayed into the simulation' interface of Cormas, they have to be written into the *init* protocol. So, select the *init* protocol of the ECEC Browser, select the #*init* method, then remove the text on the "source" panel (bottom) and write your own method on it (see previous explanation chapter 3.4.2, p. 16).

noAgent

```
"initialize the space (from poor.env file) without agent"  
self spaceModel loadEnvironmentFromFile: 'poor.env'
```

homogeneousEnv

"initialize an homogeneous space (from poor.env file) with randomly located foragers"

```
self spaceModel loadEnvironmentFromFile: 'poor.env'.
```

```
self initAgents
```

fragmentedEnv

"initialize an fragmented space (from fragmented.env file) with randomly located foragers"

```
self spaceModel loadEnvironmentFromFile: 'fragmented.env'.
```

```
self initAgents
```

Each time you want to write a new method, select another method of the desired protocol, remove the text on the "source" panel, write your own method on it and Accept. Don't be troubled, the previous method is still available. That the reason why the default #init method is still present. But as it is useless, it can be removed: select it, then right click and "Remove...".

Remarque: As we always want to display the #povBiomass of the cells, we can also use:

```
self spaceModel loadEnvironmentFromFile: 'fragmented.env' withPov: #povBiomass.
```

5.3.2. Create a specific grid

We can also define the grid size programmatically, then use the #initRandomBiomass method we have already create previously to set the biomass of each cell.

homogeneousEnv2

"initialize an homogeneous space (27x27, random biomass) with randomly located foragers"

```
self spaceModel initializeRegularX: 27 Y: 27 shape: #squared nbNeighbours: #eight boundaries: #toroidal.
```

```
self theVegetationUnits do[: cell | cell initRandomBiomass]. "or: self initCells: #initRandomBiomass."
```

```
self initAgents
```

The new grid can be saved as .env file (see chapter 9.2.3, p. 44)

5.4. Write (in the "control" protocol) a step method to be executed at each time-step of simulation

If the ECEC browser is still opened, select the *control* protocol and the #step method (otherwise reopen a browser from Cormas menu: *Programming* → *The simulation organization* → *The scheduler* → *Edit steps*). Then write the following code:

step: t

"main step: activation of all the plants (Resource dynamics), then activation of the foragers (Agents dynamics)"

"Resource dynamics (because the dynamics of the plants are independants, the activation is not mixed)"

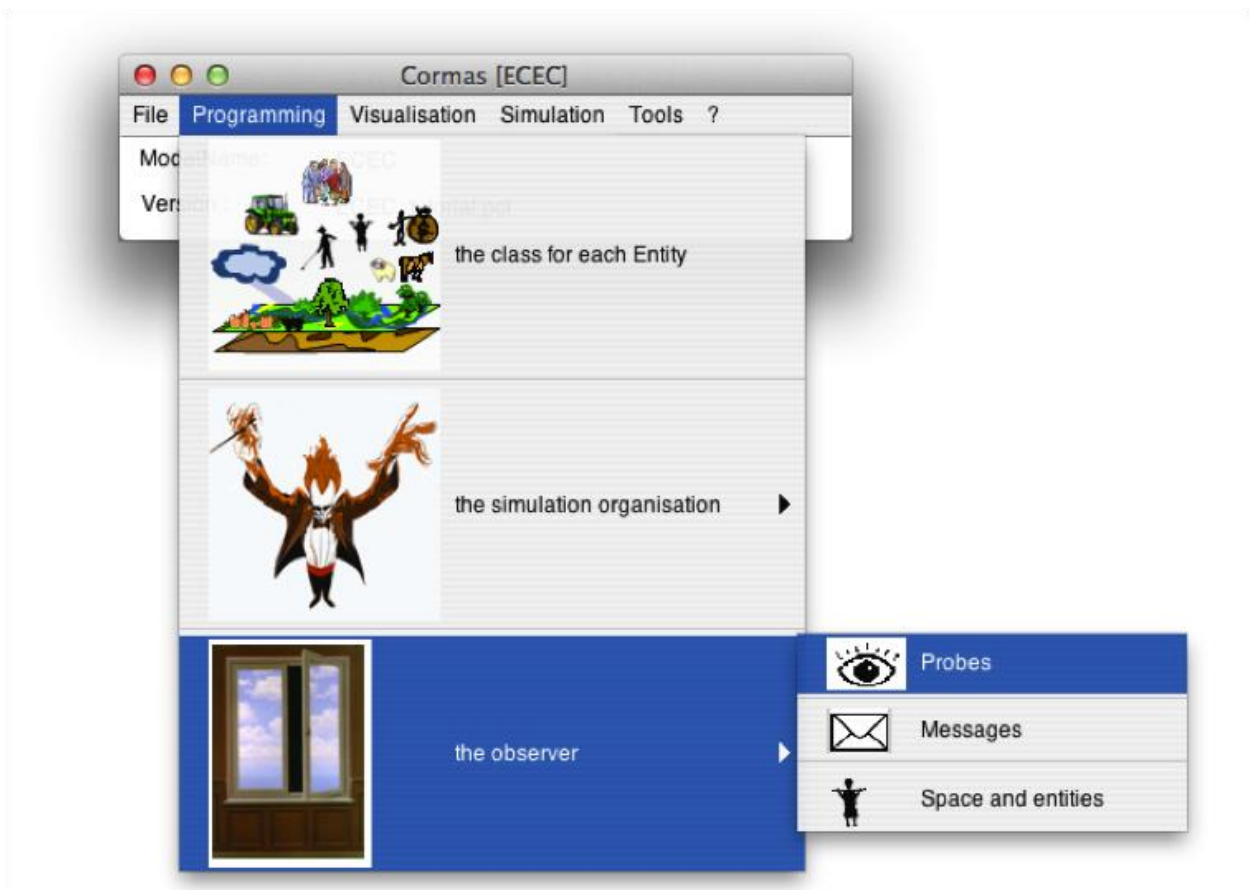
self stepEntities: self theVegetationUnits.

"Agents dynamics (because the agents may compete for plant access, the activation is randomly mixed)"

self askRandom: Forager toDo: #step

6. Designing “probes” to record the variations of markers

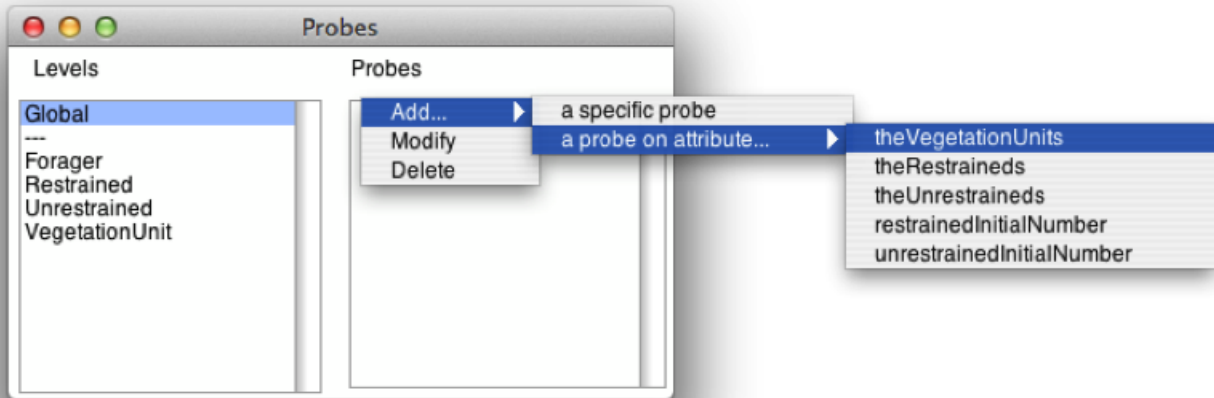
6.1. Open the “probes” definition window from the main Cormas interface



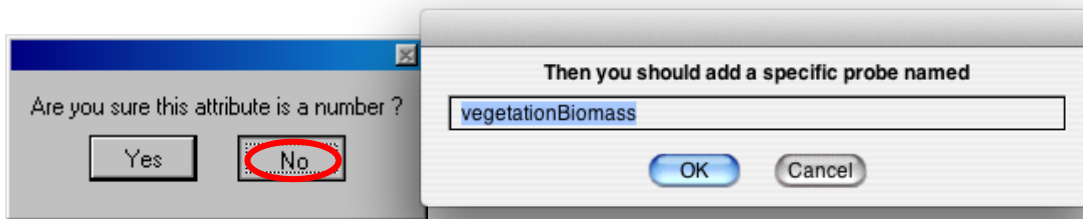
6.2. Add probes based on attributes of the model (global level)

In the “Probes” interface, select Global (as the probe aims at plotting the total biomass of the grid), right-click on the right panel to open the menu, then: *Add* → *a probe on attribute...* → *theVegetationUnits*.

Building a Cormas model from scratch step by step: the ECEC model



As *theVegetationUnits* is not a number (but a list of cellules), answer **No** to the question “Are you sure this is a number?”.



6.3. Write the code for the first one: a cumulative value over a collection

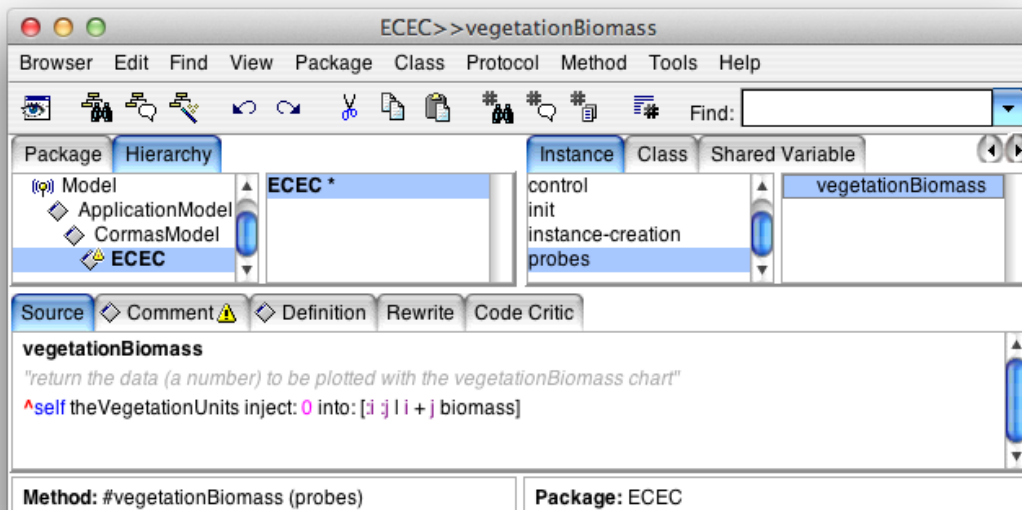
Replace the code automatically created for `#vegetationBiomass`...

`vegetationBiomass`

"modify this instruction to calculate the number to be recorded"

`^self theVegetationUnits messageX`

... with the following code:



Accept then close the ECEC browser.

6.4. Define 2 other probes: the number of restrained and unrestrained foragers

As previously, in the “Probes” interface, select Global, right-click on the right panel to open the menu, then: *Add* → *a probe on attribute...* → *theRestraineds* (or *theUnrestraineds*).

As *theRestraineds* is not a number (but a list of Restrained), answer **No** to the question “Are you sure this is a number?”, then enter *restrainedSize* as probe name. Modify the code by changing *messageX* by *size*:

restrainedSize

"return the current population size of the Restrained foragers"

^self theRestraineds size

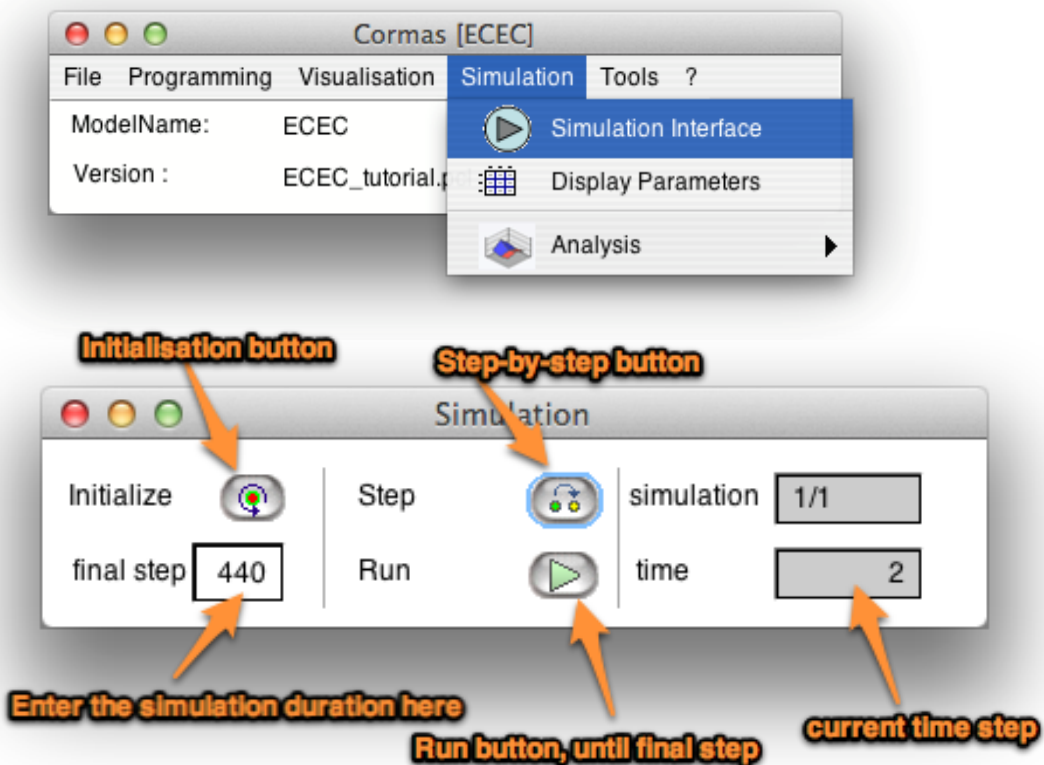
unrestrainedSize

"return the current population size of the Unrestrained foragers"

^self theUnrestraineds size

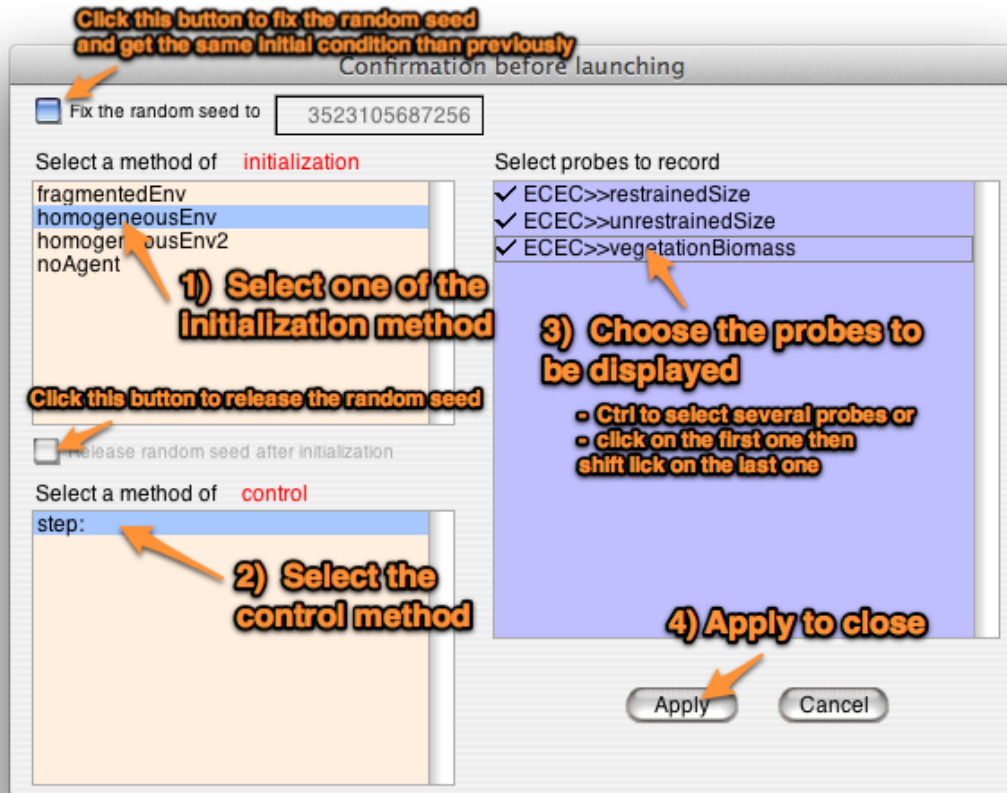
7. Simulating the model

To open the simulation interface, click on : *Simulation* → *Simulation Interface*.



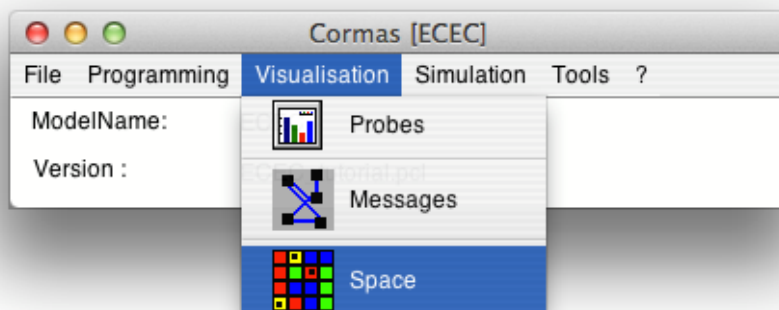
Building a Cormas model from scratch step by step: the ECEC model

Click on the “*Initialize*” button to prepare a simulation:



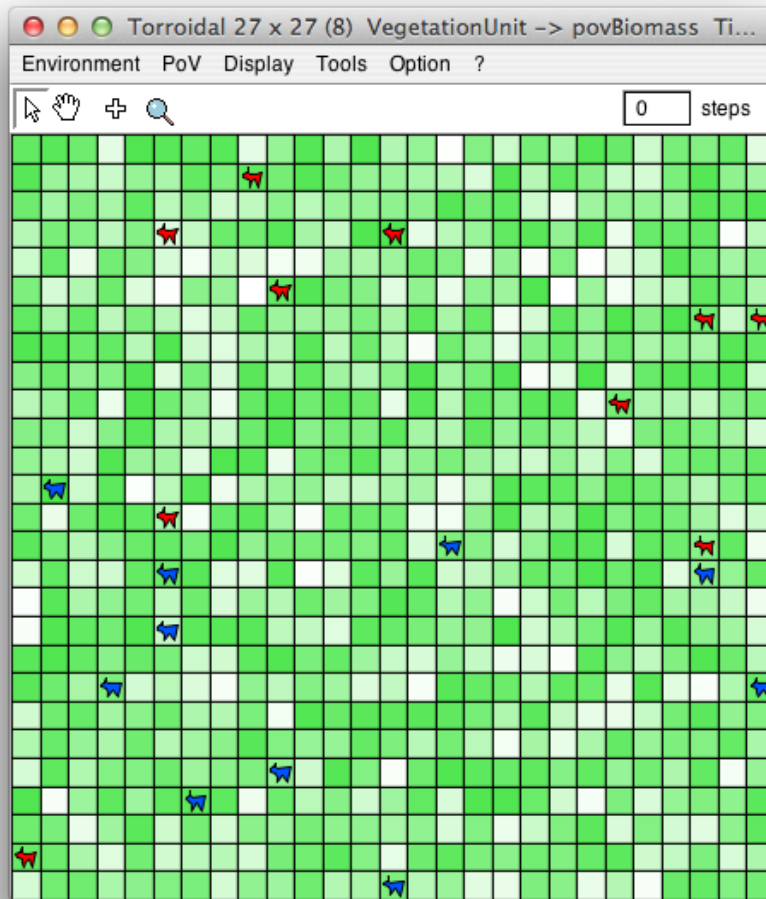
Choose one *initialization* method and one *control* method and select the probes you would like to see, then Apply.

As our model is spatialized, we may display the environment by opening the spatial grid. From Cormas menu, click on : *Visualisation* → *Space*




In order to see the vegetation state and also the foragers on the map, click on the grid menu and select *PoV* → *VegetationUnit* → *povBiomass*. Then again: *PoV* → *Forager* → *povClassName*.

Building a Cormas model from scratch step by step: the ECEC model



You can now simulate by clicking on the “Step” button or on the “Run” button once you have entered a duration value.



In order to see the resulting curves of the simulation, click on *Probes*  from *Cormas menu* → *Visualisation* → *Probes*. Then click on the probe you want to see. If you want to see several probes at once, use the Ctrl key.

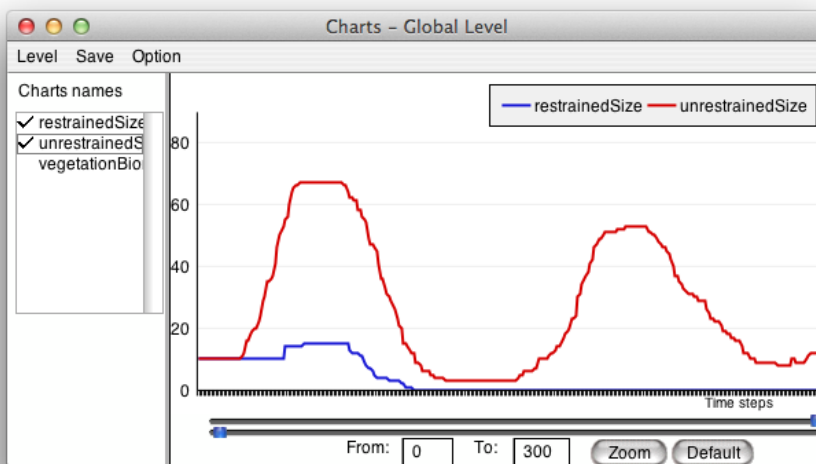
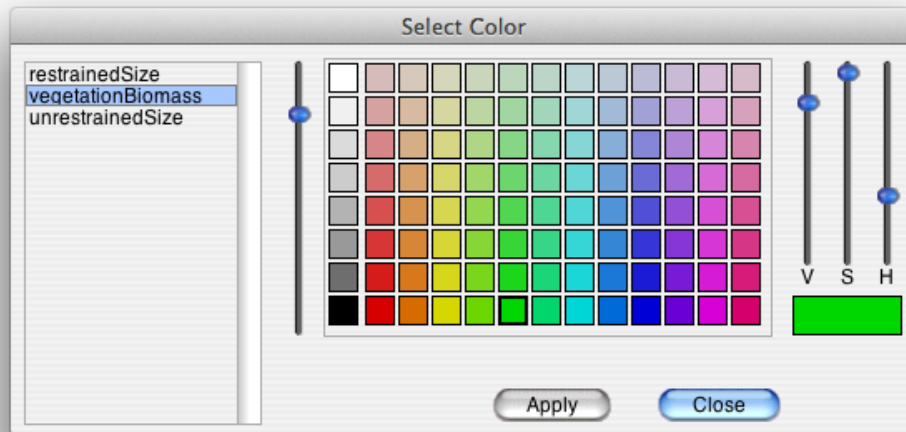


Figure 13: The first ECEC output

Building a Cormas model from scratch step by step: the ECEC model

The color of the curves may be changed to be closer to the colors of the entities. For that, right-click on the “Chart names” list or *menu* → *option* → *change line color...* Then we can change the color with the color chooser:

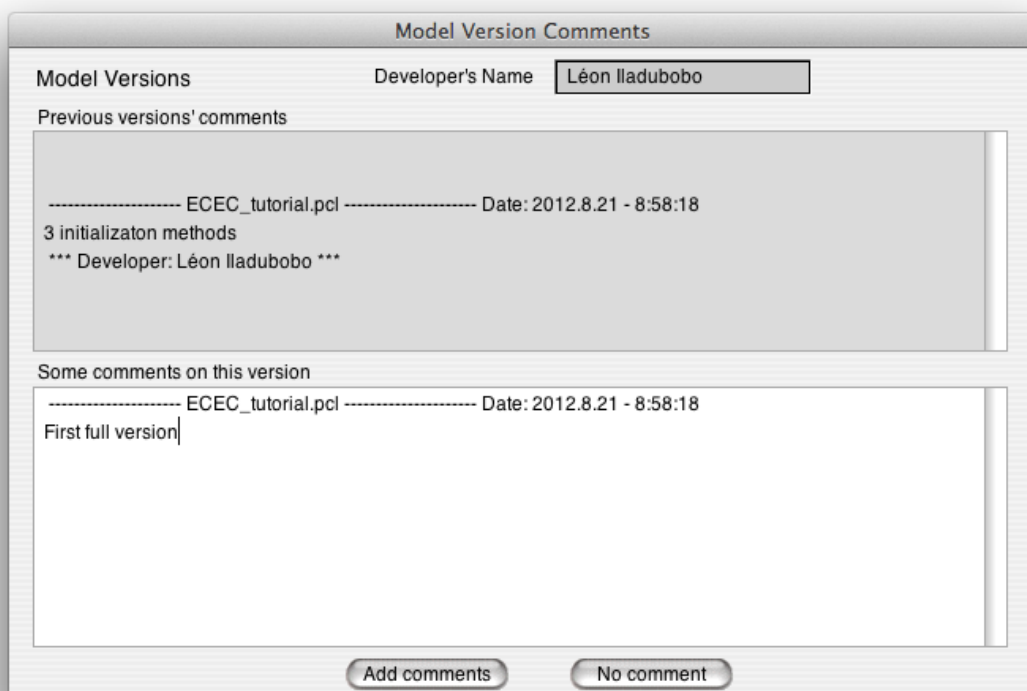


Note that the color settings are saved into ECEC for next visualizations.

8. Saving, loading and versioning the model

8.1. Saving and versioning

It is important to save regularly your model, as anything may happen... On Cormas main menu, select: *File* → *Save*. Save it as ECEC.pcl (as proposed), but you can also choose another name if you want to have several versions of your model. Before to save, Cormas will ask if you want to add some comments of this version:

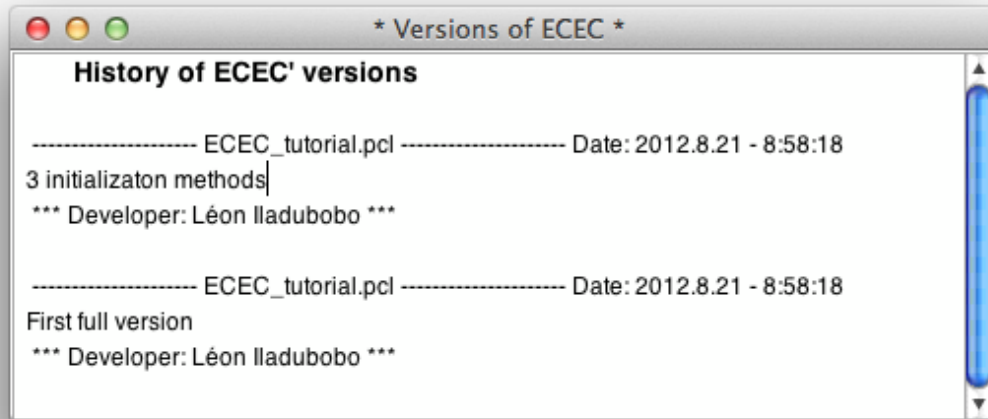


Building a Cormas model from scratch step by step: the ECEC model

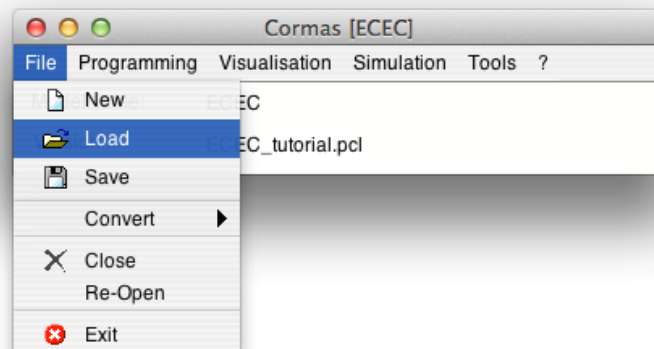
Enter some comments (like “First full version” for example) and click on “Add Comment” button.

The Saving procedure has automatically creates an “ECEC” folder, under “cormas/models”, and the model is saved on the disk as a file with .PCL extension (Package). You may also use the old process of saving models with .ST file. For that, select *File* → *Convert* → *Save as ST*

You can see the history of your model by selecting the Cormas menu: *Tools* → *Display model' versions*



8.2. Loading the model



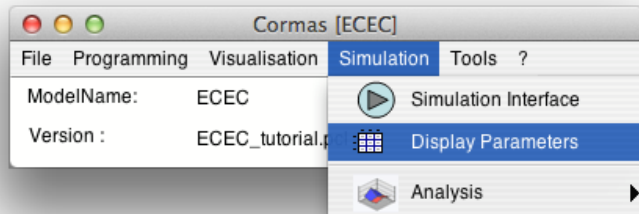
Then choose the version you want to load.

9. Playing with the model

Cormas has also been improved in order to facilitate the interactions with the simulation. The purpose is to easily modify the values of the parameters, to manipulate the agents on the grid and even to modify their behavior without coding.

9.1. Changing the parameters' values

With this new version, it is simpler to change the value of a parameter. All parameters can be displayed in an interactive table. To open it, click on *Cormas menu* → *Simulation* → *Display Parameters*:



The table only shows the simple parameters (called literals) of numerical, string or boolean types:

The screenshot shows the 'Parameters' dialog box with a table of parameters. The table has columns for 'N°', 'Class', 'Attribute', and 'Value'. The first row is selected, and the value '10' is highlighted in the 'Value' column.

N°	Class	Attribute	Value
1	ECEC	restrainedInitialNumber	10
2	ECEC	unrestrainedInitialNumber	10
3	VegetationUnit class	r	0.2
4	VegetationUnit class	K	10
5	VegetationUnit	biomass	0
6	Restrained class	fertilityThreshold	100
7	Restrained class	harvestRate	0.5
8	Restrained class	catabolicRate	2
9	Restrained	energy	50
10	Unrestrained class	fertilityThreshold	100
11	Unrestrained class	harvestRate	0.99
12	Unrestrained class	catabolicRate	2
13	Unrestrained	energy	50

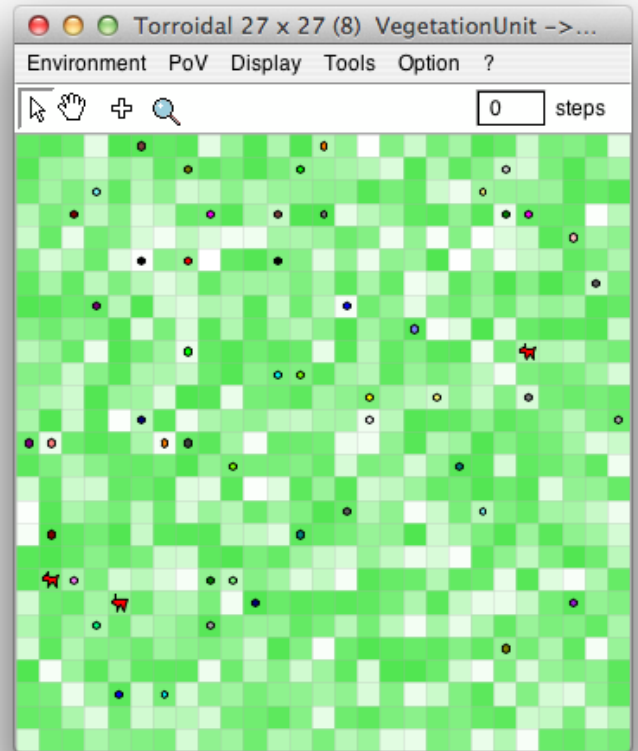
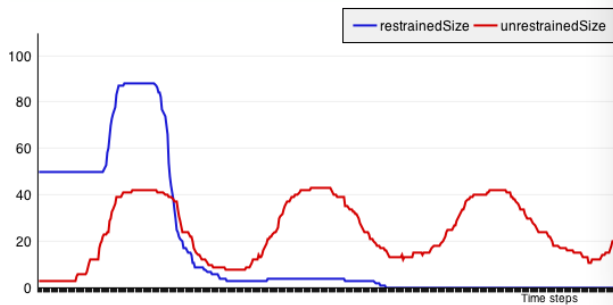
To change a value, select it, enter the new value and click on “*Apply new values*” button. For example, to start a simulation with **50 Restrained** and **3 Unrestrained** foragers, enter 50 and 3 respectively in the *restrainedInitialNumber* and *unrestrainedInitialNumber* cells, then “Apply”.

Building a Cormas model from scratch step by step: the ECEC model

When initializing the simulation, 53 agents are created.

To facilitate the visualization, the points of view of the Restrained is set to *povId* and to *povClassName* for the Unrestrained.

Note that in terms of result, this new initial state does not prevent the population of restrained foragers to collapse.



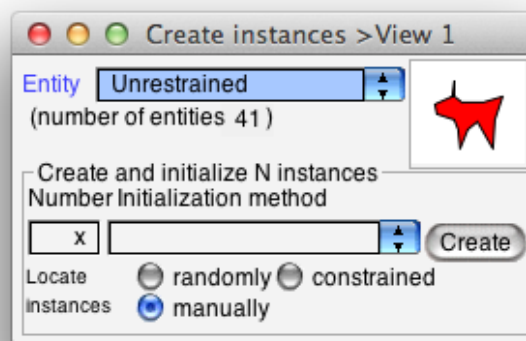
If you run another simulation, the modified parameters will be used again (i.e. 50 Restrained and 3 Unrestrained). To come back to the default values (10 and 10), click on the “*Back to the Default Values*” button of the Parameters interface.

But if the modified parameters seem better for future use of the model, you can save them by clicking on the “*Save as Default Values*” button. Thus, the code of the model will be modified to store these new values.

9.2. Manipulating the agents on the grid

9.2.1. Creating new foragers

By clicking on the + button of the grid, you can instantiate new agents.




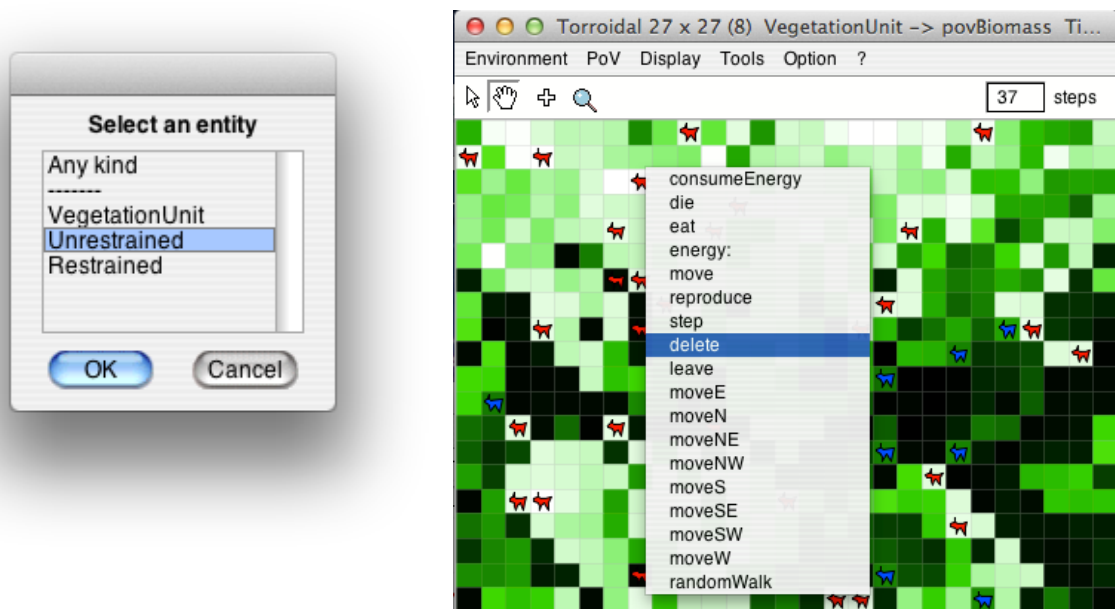
Building a Cormas model from scratch step by step: the ECEC model

Set the entity type to be created and click on “*Create*” button. Then, each time you will click on the grid, a new forager will be instantiated on the target cell.

If you want to limit the number of new foragers, enter the limit number on the “X” input box before to click on “*Create*”.

9.2.2. Moving and sending operation to the agents

By clicking on the hand button of the grid , you can manipulate the agents. Select the entity type to manipulate and OK.

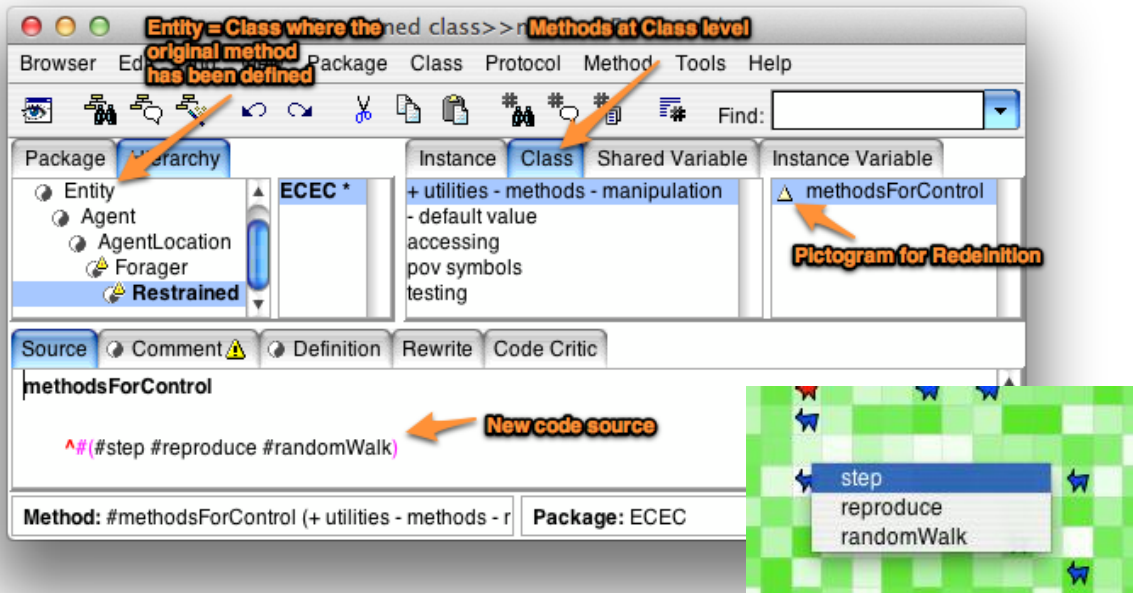


Thus, by right-clicking on a forager, he will perform the operation that you have select from the contextual menu. You can also move this forager where you want with a drag and drop. If “Any kind” have been selected, then you can move any kind of located entity or send a method to this entity (the contextual menu adapts itself to the clicked figure).

To be displayed on the contextual menu, Cormas lists all the methods of your model excluding those of ‘pov’, ‘probes’ and ‘init’ protocol. It also add to this list the methods of the super-class that belong to the protocols starting with ‘* ...’.

But you may customize your own contextual menu, by redefining the #methodsForControl (at class level). This method can be overwritten as following:

Building a Cormas model from scratch step by step: the ECEC model

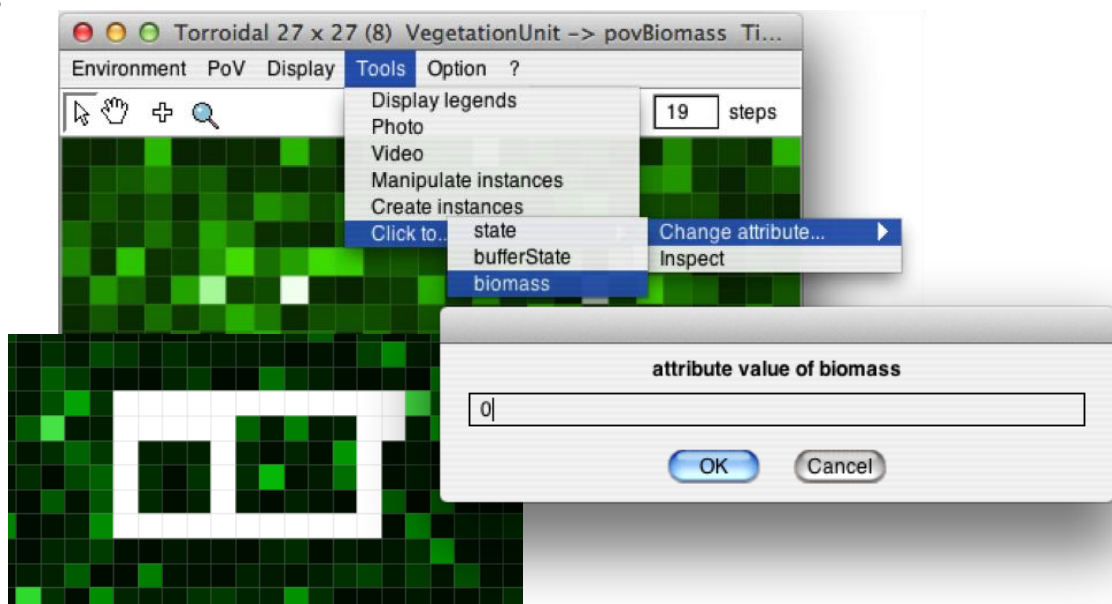


9.2.3. Changing the environment

From the grid menu, you can:

- set the environment as torroidal or closed grid: *Environment* → *Modify* → *Grid boundaries* → *Closed*
- change the shape of the cells (and their connectivity): *Environment* → *Modify* → *Cell shape* → *Hexagonal*

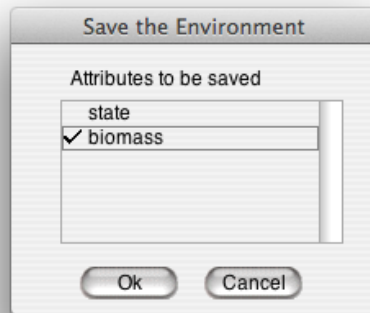
You can also set a new spatial configuration by changing the *biomass* attribute value and designing with the mouse. From the grid menu, *Tools* → *Click to...* → *Change attribute* → *biomass*




Then enter 0 and click on the cells (or drag).

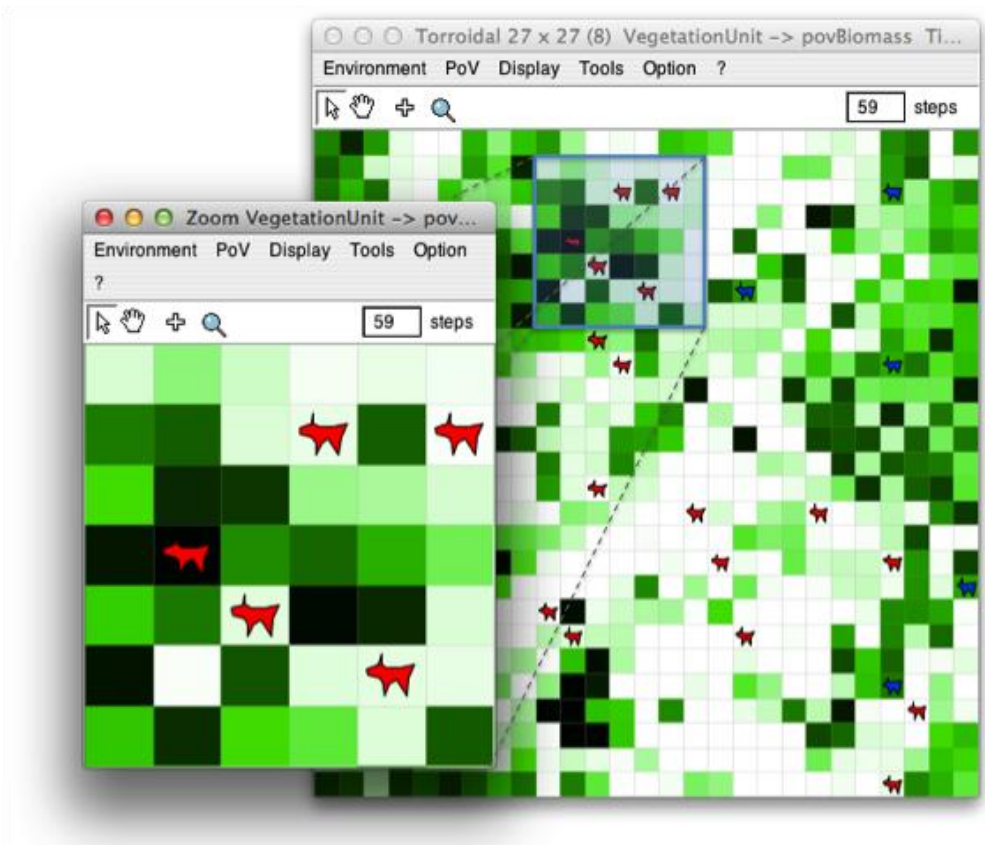
Building a Cormas model from scratch step by step: the ECEC model

Thus you can save this new environment for future simulations: *grid menu* → *Environment* → *Save* → *Env type*, select *biomass* and enter the fine name.



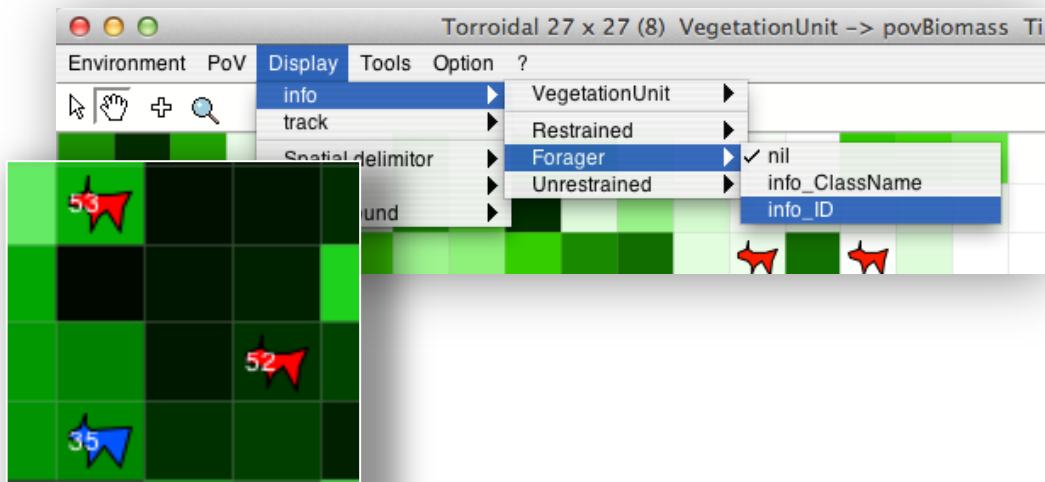
9.2.4. Other display options

Zoom: You can zoom on a part of the grid by selecting the 4th button , then by doing a drag on the region of the grid you want to zoom in.



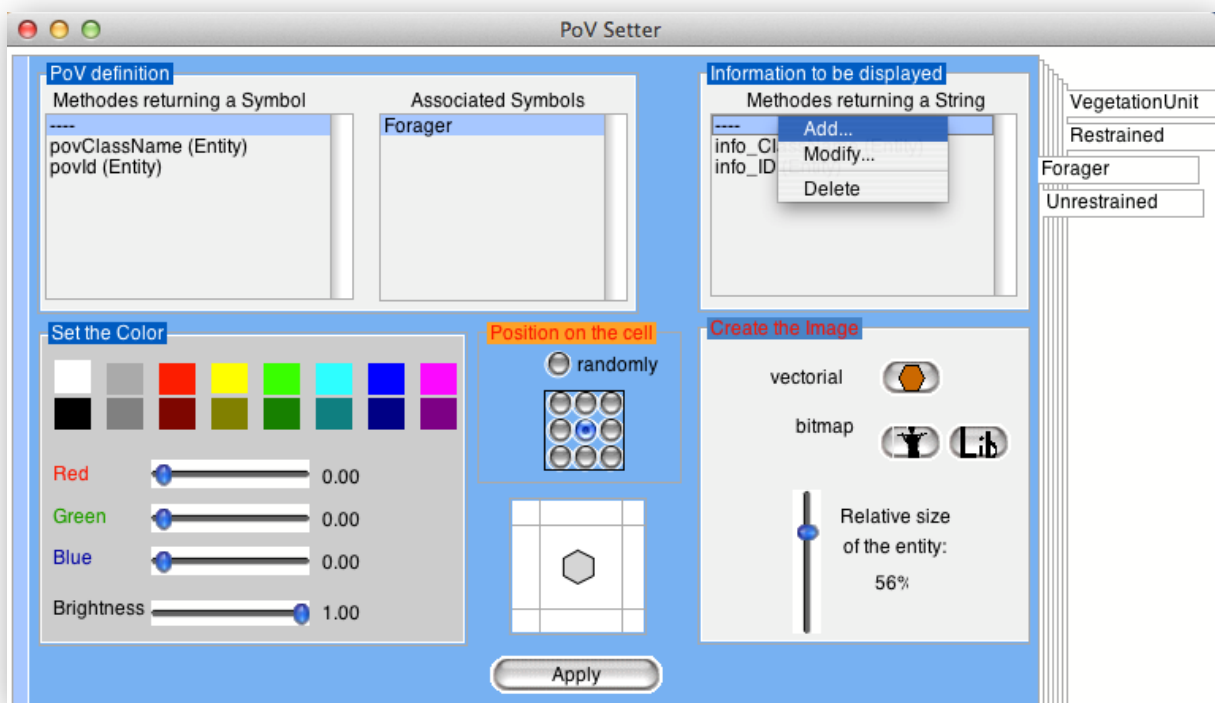
Display information: You can display some information near each entity by selecting the menu *Display* → *info* → *Forager* → *info_ID* (for example):

Building a Cormas model from scratch step by step: the ECEC model



This option is also available from the contextual menu of an entity.

The default information available is the name of the class and the identifier (id) of the forager. But this is easily add new information such as the energy level of each agent. For that, open the PoV Setter (see 4.1.1, p. 23), select 'Forager' item, right click on the 'information' panel on top right and click on 'Add...'



Enter *info_energy* as name:



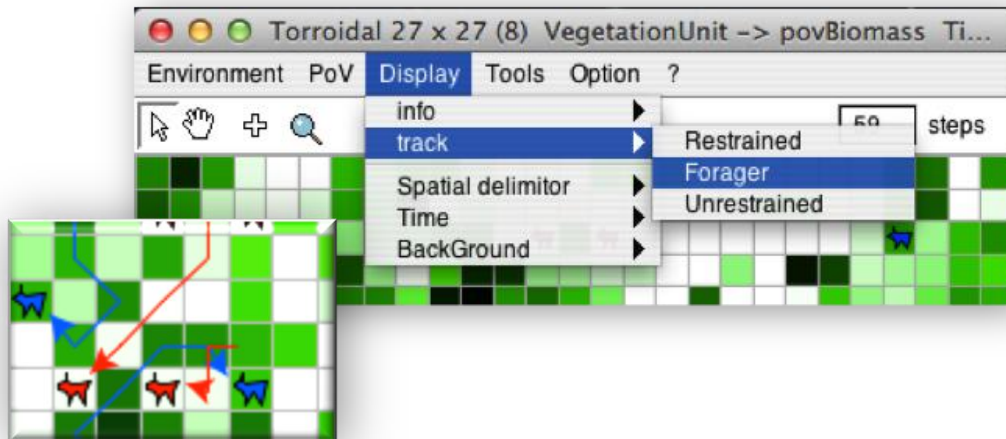
Building a Cormas model from scratch step by step: the ECEC model

Then change the code with:

```
info_energy
"return a String "
^self energy rounded printString
```

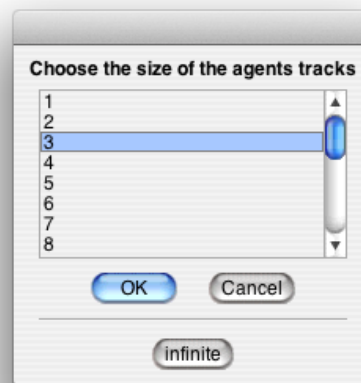
(As the value of energy may be a long float, it is better to round it for display facility)

Tracking: You can track the foragers by selecting the menu *Display* → *track* → *Forager* :



This option is also available from the contextual menu of an entity.

You can set the length of the track from the grid menu: *Option* → *set track length*, then by choosing the length

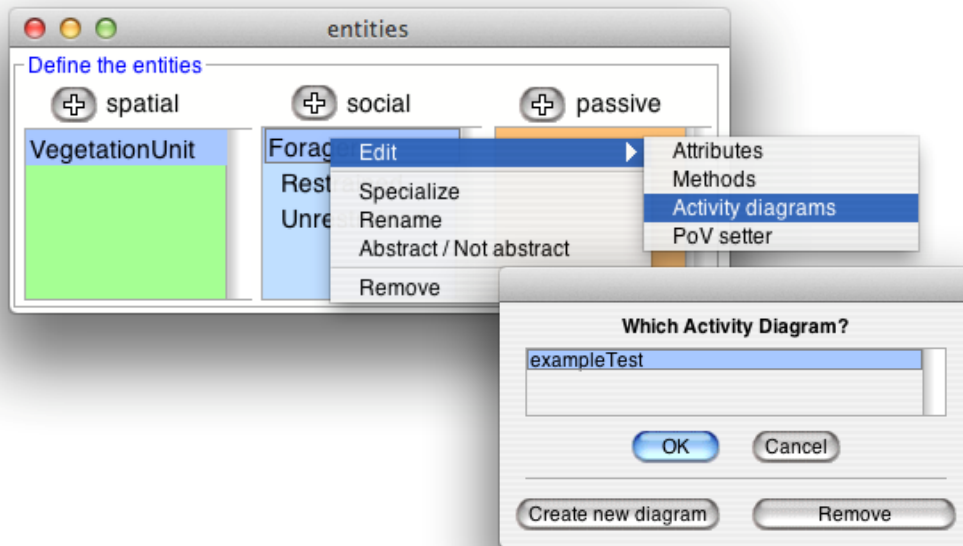


9.3. Executable activity diagram editor to modify the agents' behavior

A new tool was created that enables the drawing of simple activity diagrams and to execute them without any need for translation into code. Indeed, this diagram editor allows the creation of new activity diagrams (or re-opening formers) that are interpreted “on the fly” by Cormas. Users can modify the simulator while it is running, without stopping or restarting the simulation.

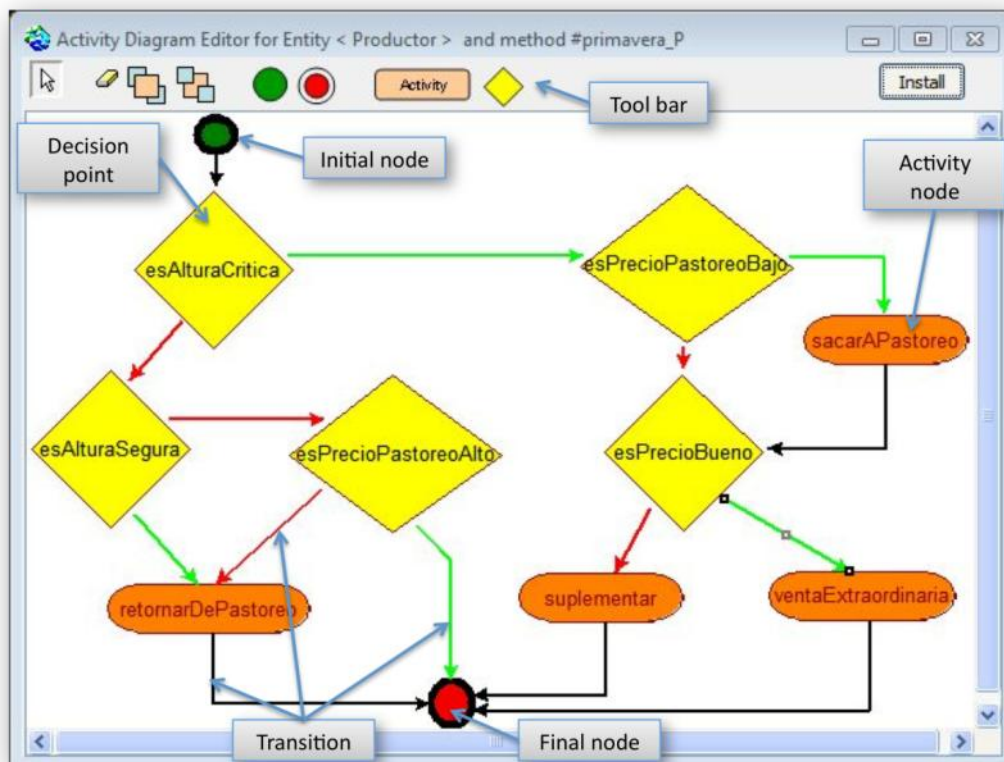
Building a Cormas model from scratch step by step: the ECEC model

To create a new diagram, right click on Forager class from “Entities” interface



Then click on “Create new diagram” button:

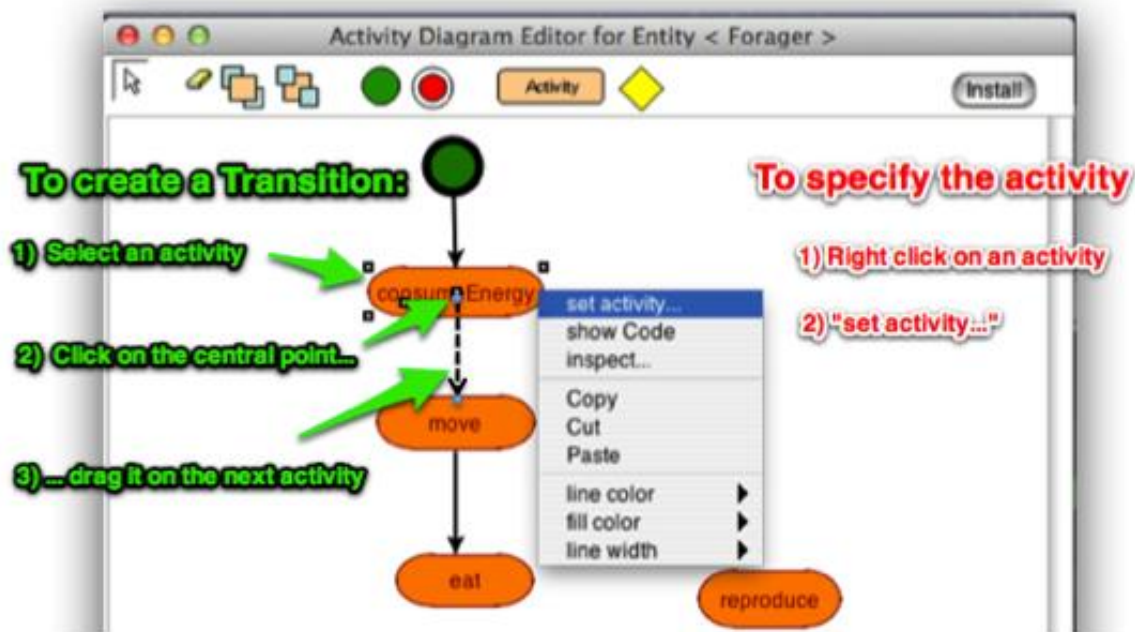
For simplicity sake and user friendliness, the elements available on the diagram editor are restricted to initial and final nodes, simple activity nodes (without parameters nor ability to handle an activity output), transitions and decision points. It does not include more sophisticated features such as swimlane, iteration and concurrency notations. The purpose of this executable editor is to facilitate understanding and usability so that non-computer engineers may use it.



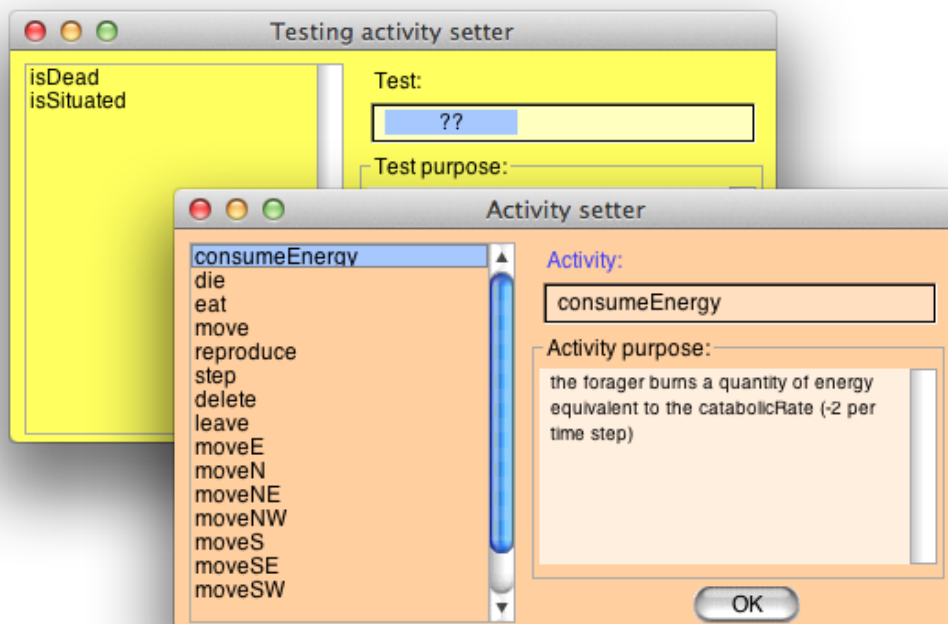
Building a Cormas model from scratch step by step: the ECEC model

The decision points do not comply exactly with UML notation but are rather like the old flow chart diagrams for which the question is written into the diamond and from which only two transitions emerge indicating the fulfillment (true) or the negative answer (false).

By selecting an activity node or a decision point on the tool bar, you can add a new element on the diagram. Thereafter, you can choose the operation to be performed by this element. Each element proposes a drop-down menu (right click) to display an activity setter from which you may choose the method that will be associated with the selected node.



Two examples of activity setter, for activity node (orange) and for decision point (yellow):



The activity setter displays a list of methods belonging to the target class (i.e. Forager). This list is set up automatically by Cormas that inspects all the simple methods defined within the class and its super-classes². By clicking on a name, the purpose of the associated method is displayed. You may also inspect the method's code by right clicking on it.

As you can see, there is plenty methods for the activity setter (the list starts with the Forager's methods), but very few for the Decision node (`#isDead` and `#isSituating` which are generic methods of Agent). Thus, we need to add new questions for the ECEC model, such as `#isEnergyHigh` (in order to reproduce) and `#isEnergyTooLow` to test if the forager will die. These methods have to be written in the *testing* protocol in order to be display by the Decision point setter.

Open a browser on the Forager class (double click on Forager in the “entities” interface), and create a new protocol called “testing” (see Figure 11, p. 17). Then write the follow methods:

isEnergyHigh

"Tests if energy is upper than the fertilityThreshold (100), in order to reproduce"

```
^self energy >= self class fertilityThreshold
```

isEnergyTooLow

"Tests if energy is 0 or less, so that the forager will die"

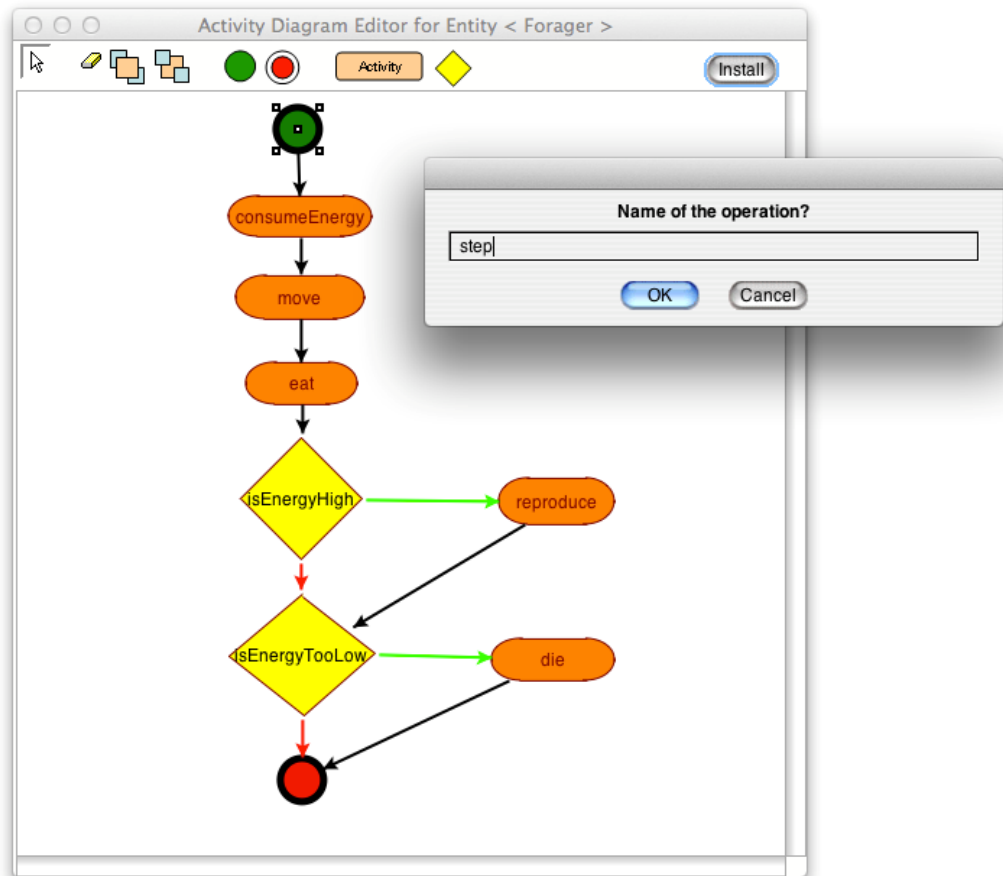
```
^self energy <= 0
```

Now, you can finalize the activity diagram. Obviously, from a decision point, you will create two transitions: one for which the answer is *true* (green) and one for *false* (red).

When it is done, you must save it by clicking on the “Install” button. Enter *step* as the name of the operation. The original `#step` method (previously defined, see chapter 3.9, Coding the “step” method of the forager agent, p. 23) will be overwritten.

² Thanks to Smalltalk code organization in protocols (also called “categories” for which the included methods share a close semantic), the methods of a decision point are collected by Cormas when inspecting the “testing” protocol of the target class and its super classes.

Building a Cormas model from scratch step by step: the ECEC model



The design is incremental: saving a new diagram generates a new method of the agent that is immediately available and can be called in turn (future activity setter will display this new method name). A right-click on an activity or a decision point opens either a code editor targeting the selected operation, or another diagram editor displaying the previously saved activities.

When saving the new diagram (“Install” button), Cormas checks if it is coherent, then generates two operations in the target class:

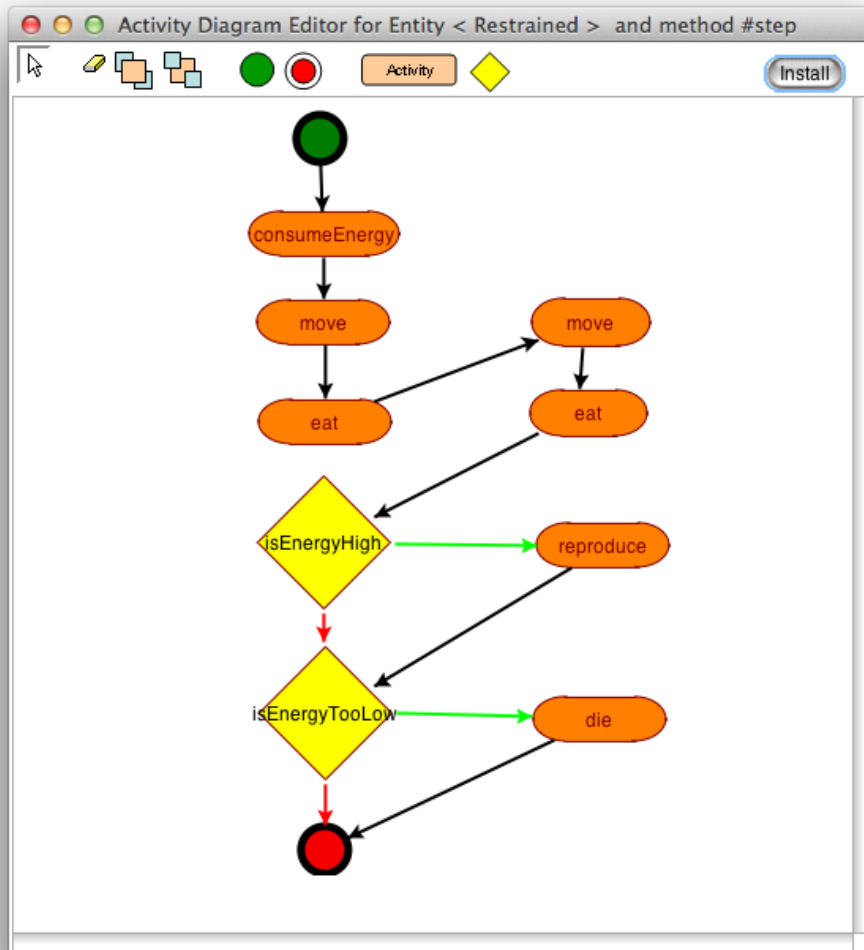
- one to store the diagram: #activity_step, defined at the class level
- and one to execute it (the new #step method):

```
step  
"Forager openActivityDiagram: #step "  
"This method was automatically generated by the ActivityDiagramEditor."  
self performActivityDiagram: #step
```

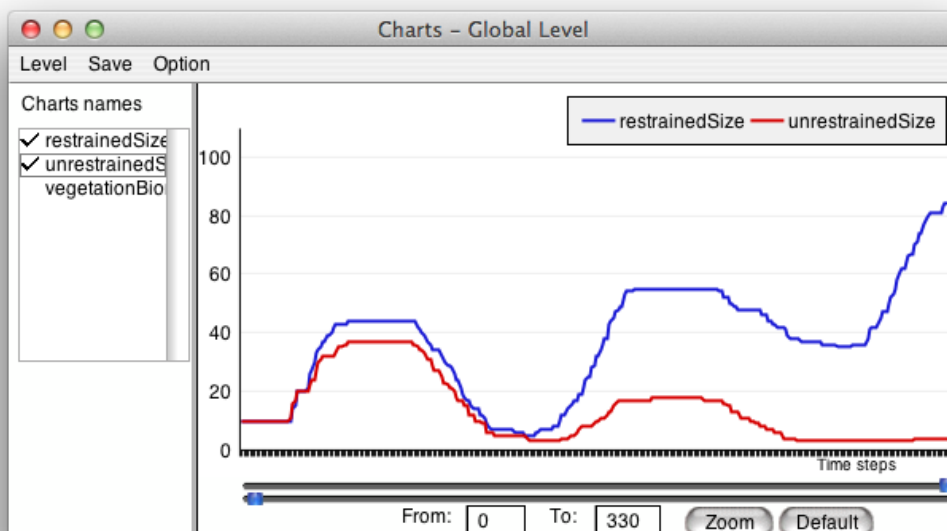
An activity diagram is not compiled into programming language but is directly interpreted by the simulation, without an intermediate stage. In other words, the new activity diagram is saved as part of the source code of the ABM. As it belongs to the executable code, it can be reopened at anytime, modified and performed without the requirement of developer skills.

Building a Cormas model from scratch step by step: the ECEC model

Thus, from basic operations already defined by the modeler, anyone may generate new upper level behavior without any programming skills. For example, you can modify the behavior of the Restrained foragers: open an editor by right clicking on Restrained class from “Entities” interface and select *Activity diagram*. Choose the *step* diagram and modify it as following:



Thus by running a simulation, we observe that the Restrained population does not collapse any more.



To remove the modified diagram, right click on Restrained class from “*Entities*” interface, then *select Edit* → *Activity Diagram*, then click on the “Remove” button.

You can save the model because this is its last version for the scope of this tutorial.

10. Analyzing the model

We present now the ways to conduct *sensitivity analysis* with Cormas. Analyzing is an important part of the modeling activity and should not be forgotten. It helps to reveal and correct bugs and points of failure, but above all, it provides to the one who conducts this analysis, a better understanding of the behavior of the model. As Jean Piaget said, “we only know an object when acting on it and transforming it”.

Choice of indicators (probes): An important starting point is the choice of the probes to collect. Indeed, one can study an ABM at two distinct levels of analysis: either globally, giving for example the number of agents in a population or the date of collapse, or at the individual level (energy of an agent). Generally, sensitivity analyzes are limited to the global level. This seems already sufficient since it needs a lot of time.

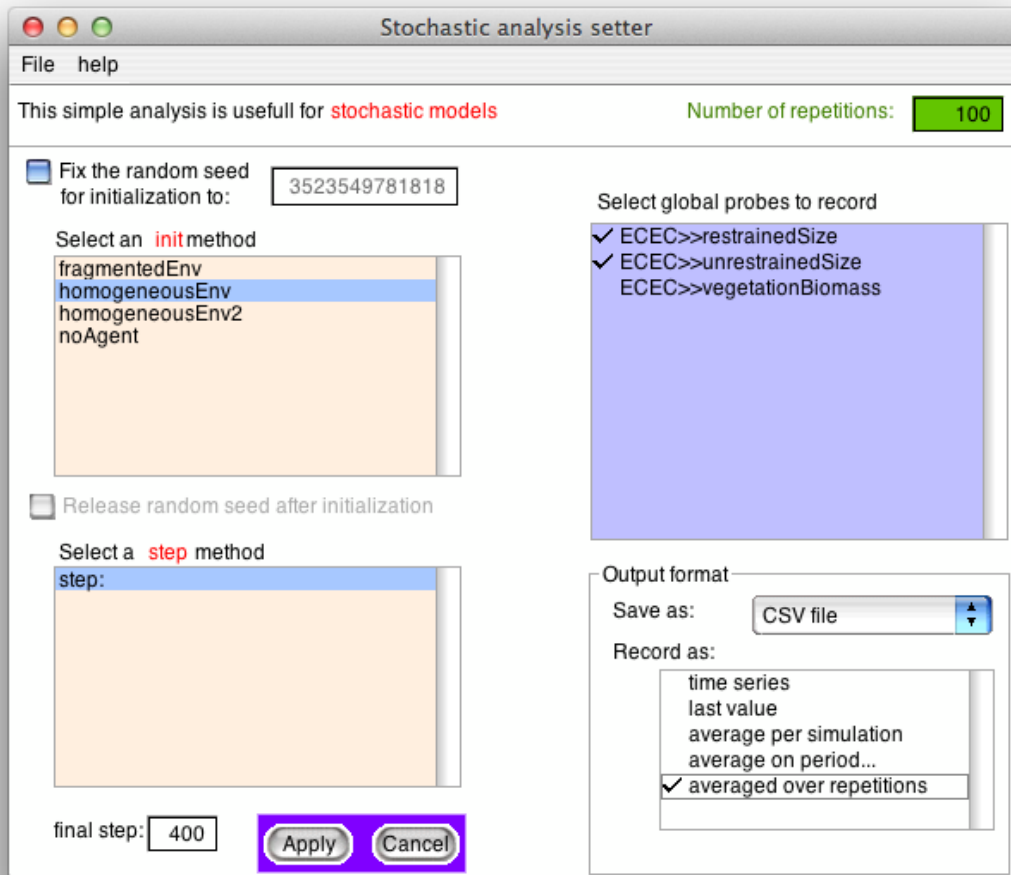
The studied indicator depends also on how it is collected. For example, recording the average of a probe over the whole simulation can hide important information about the sensitivity of the model. For the present case, the previous simulations of ECEC show that there is often a first crisis for both populations of foragers. After that, one population often disappears while the other one is fluctuating. So rather to record the size of both populations at time 400 for example, it is better to record the average size between step 150 (after the first crisis) and 400.

Cormas don't propose analyze tools; it just allows to launch many simulations according to parameters values and to export the data of the simulations in different file formats: CSV (comma-separated values where the data (numbers and text) are stored in plain-text form) or Excel files. You can then analyze your data with your favorite software...

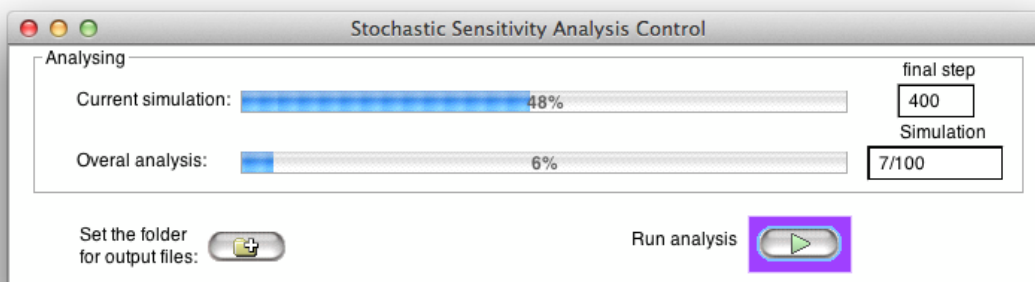
10.1. Simple stochastic analysis

As ECEC is a stochastic model (there is randomness for initialization, for mixing the agent order for activation and for the #move process), we can conduct a simple analysis. On Cormas main menu, select: *Simulation* → *Analysis* → *Simple Stochastic Analysis*:

Building a Cormas model from scratch step by step: the ECEC model



Select the output format of the files: CSV or Excel (Be careful, on Mac, the connection of VW with Excel doesn't work, and MSAccess is not ready) and record as “*averaged over repetitions*”. Then “Apply”.



On the new interface that pops up, set “*the folder for output files*” by clicking on the appropriate button. By default, Cormas proposes a folder name starting with the analysis type and the current day. The folder will be created into ECEC/data/. When ok, click on “*Run analysis*” button. To speed up the simulation, it is recommended to close the grid displaying the agents and the probes' interface displaying the curves. 3 files will be created:

- One (*analysisSettings.csv*) containing the settings of the analysis: you can run this analysis again by reopening *Simple Stochastic Analysis* interface and select *File* → *Load analysis settings*.

- And two (*AveragedTimeSeries_unrestrainedSize.csv* and *AveragedTimeSeries_restrainedSize.csv*) containing the data of the analysis.

Thus, into Excel (or other spreadsheet application), you get the following graphic for which the curves are smoother than for one simulation (see first output graph, p. 38). This is due to the “*averaged over repetitions*” format chosen for the output: Cormas calculates the averaged probe (restrained size for instance) by taking for each step the average value of over the 100 repetitions.

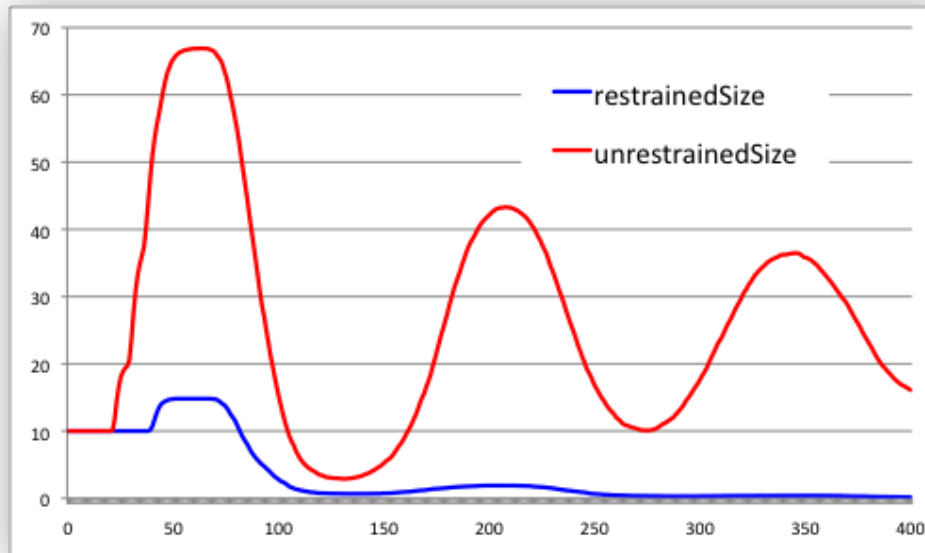


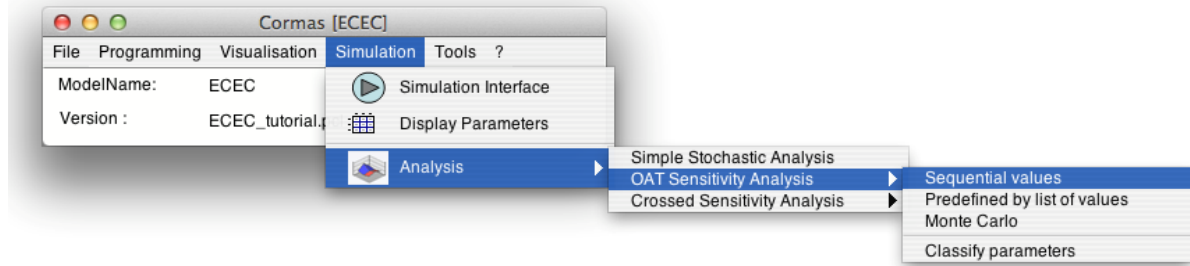
Figure 14: ECEC outputs averaged over 100 repetitions

10.2. Individual signature of parameters

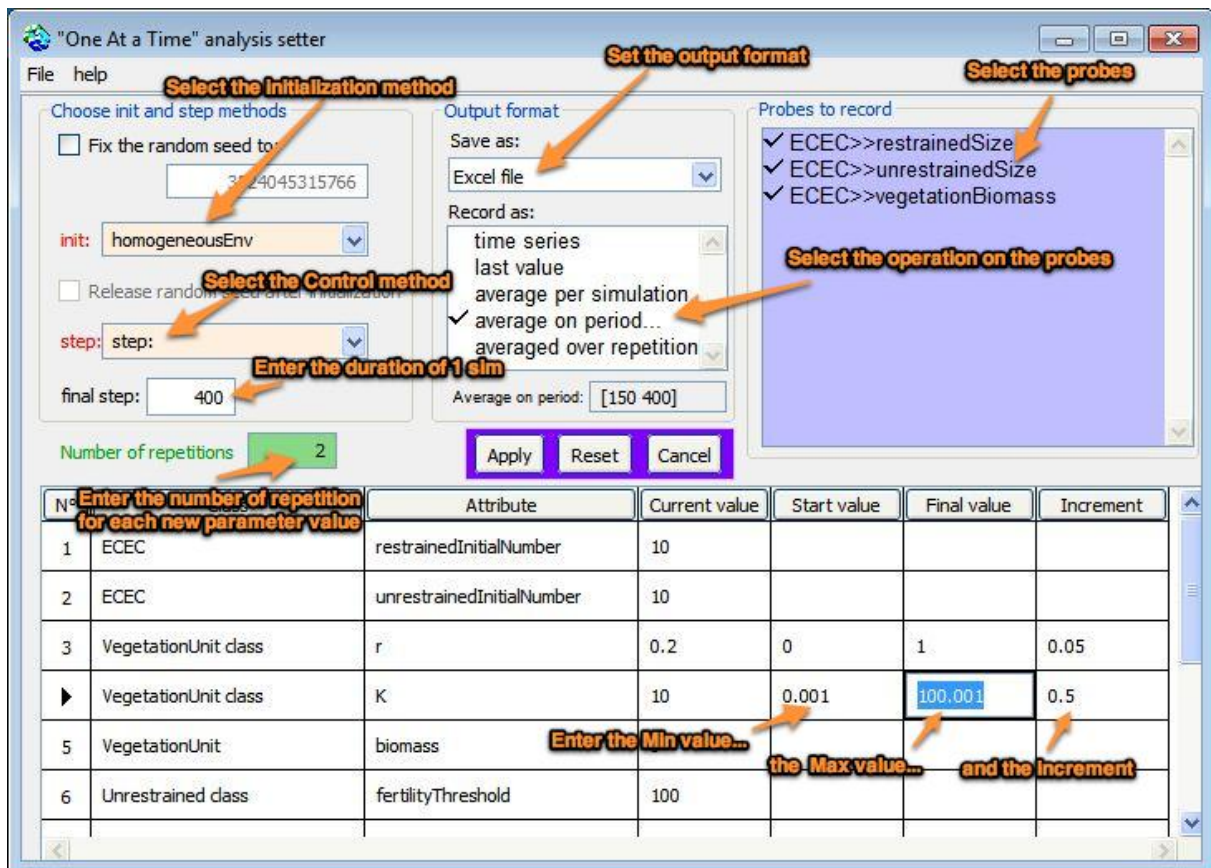
The individual signature of a parameter is useful to know its influence on an indicator (or a probe). By fixing the other parameters at their reference value, we collect the outputs of the selected probes by changing for each simulation the value of one parameter. For these analyzes, it is best to choose a wide range of variation. This gives a graph showing the output values according to the parameter whose sensitivity is given by the slope of the curve. A straight line, parallel to the X axis (the parameter axis) indicates a lack of sensitivity of the indicator to this parameter. For stochastic models, simulations must be repeated for each value of the parameter. Statistically, the smaller the increment step is, the less we need to repeat the simulations. Thus, if you decide to run 100 simulations, it is best to conduct the analysis of 50 parameter values, repeating only twice each value, rather than the reverse.

To conduct such OAT analysis (One factor At a Time), you can use either “Sequential values”, “Predefined by list of values” or “Monte Carlo”:

Building a Cormas model from scratch step by step: the ECEC model



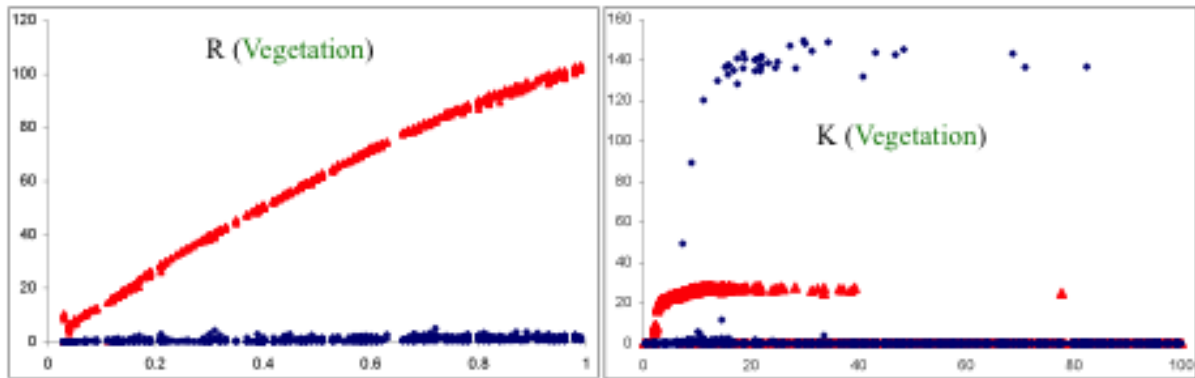
Let's say we will use a OAT Sequential values analysis.



Select the *init* and *control* methods (#homogeneousEnv and #step:). Enter the final step (400), the number of repetitions (2, for example), select the output file format (do not use Excel for Mac), the format of the probes (or example, 'average on period...' [150 to 400] and the probes (*restrainedSize*, *unrestrainedSize* and *vegetationBiomass*). Then enter the values of the interval for each parameter to be analysed. Example: the attribute *r* of VegetationUnit can be analyzed from 0 ('Start value') to 1 ('Final value') with an 'Increment' step of 0.05.

The following figures show some results of this kind of analysis:

Building a Cormas model from scratch step by step: the ECEC model



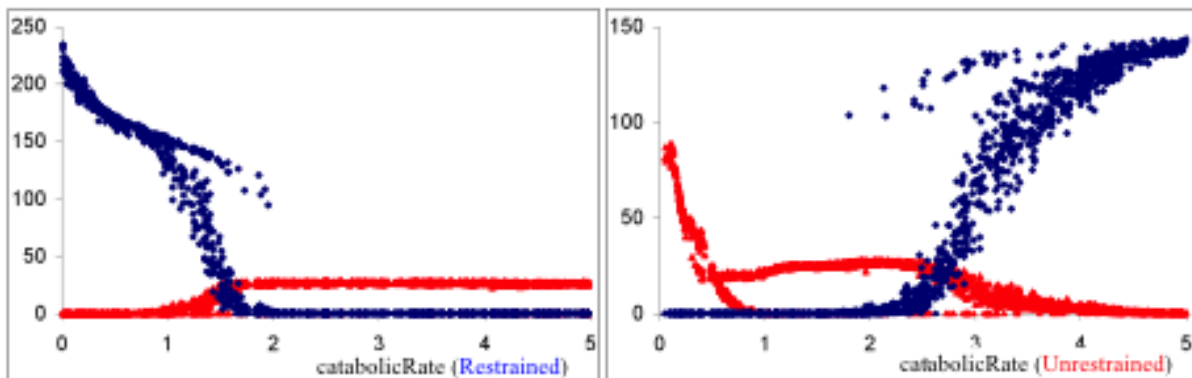
In presence of Unrestrained, the parameter r does not affect the dynamics of the Restrained. But the size of the population of Unrestrained increases almost linearly with r .

Population responses to the K parameter is a bit more complicated:

When K is smaller than 5, there is a rapid increase in Unrestrained size at and systematic exclusion of small.

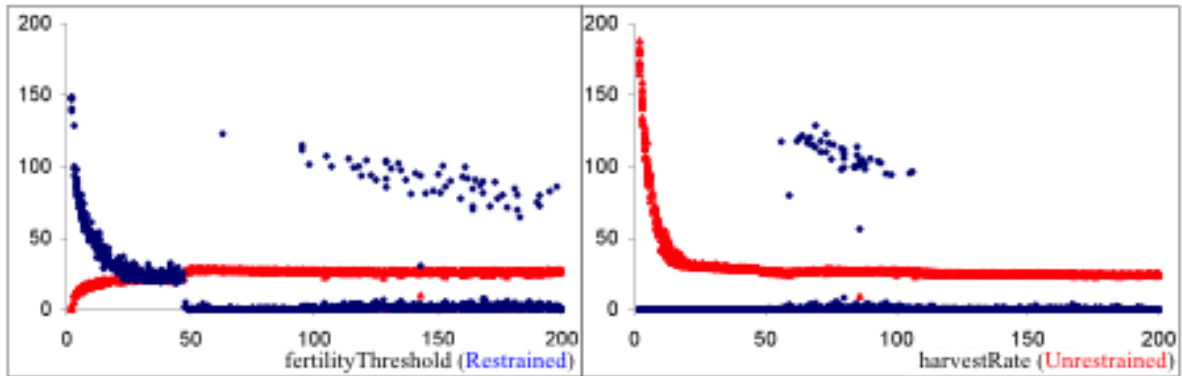
When K is in the range $[5, 40]$, the average size of Unrestrained stabilized at around 23 agents. But when K tends towards the upper part of this interval, the collapse of Unrestrained is frequent. For these values, the Restrained are regularly excluded, but for some few cases, they may survive, with an increase of the population size up to 140 agents.

Beyond 40, both populations always collapse. Despite similar vegetation dynamics at the beginning of the simulation (similar rate of increase in early growth then slowdown for small values of K), the environment reached a first peak of biomass, which varies according to K (3000 or 6000 when $K = 10$ or 40). The height of the first peak allows a rapid increase in population. When this peak is moderately high, the population size remains modest (65 Unrestraineds, $K = 10$). But if this peak is high, the population grows quickly (up to 200 agents for $K = 40$). In this case, the first crisis is very violent and leads to extinction of the populations.

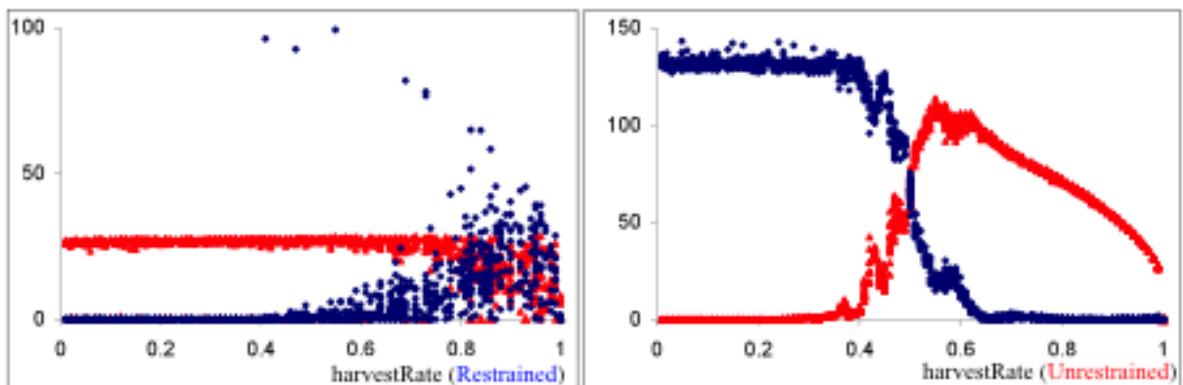


Here also the model answers are not simple. A catabolic rate of the Restrained higher than 2 (baseline) causes the systematic exclusion of the Restrained. But for values lower than 2, these agents become competitive. For a rate close to 1.5, both populations coexist frequently. Then, the lower this rate is, the more the Restrained population is high. In this case, they

reduce the resources for the Unrestrained who are systematically excluded when the rate is below 1.



The signature of the Restrained' threshold fertility shows two very distinct phases. Above 48, the Restrained are excluded and they do not affect the Unrestrained. For threshold values above 100, there is nevertheless few cases of non-exclusion: Restrained agents do not consume energy to reproduce then can more easily survive. But this stable situation suddenly changes when the threshold fertility falls below 48. Suddenly the Restrained become competitive and rapidly invade the space. With an increase of the population size, the Restrained are struggling the Unrestrained.



The higher the harvest rate of the Restrained (initially set at 50%) is close to that of the Unrestrained, the more the conflict between both populations increases with the balance of the forces.

The signature of Unrestrained's harvest rate is more original. For values greater than 40%, the size of Unrestrained increases. But from a maximum rate of 56%, the size of this population gradually decreases. Compared to the Restrained harvest rate (50%), it is better thus to have a rate just higher (between 53% and 65%), because upper rates reduce the size of their population. So much that for very high levels (close to the baseline, 99%), the population may collapse by itself (the only case of elimination of oth populations happens for very high rates).

10.3. Sorting the sensitivity of the model to parameters

This analysis seeks to describe the impact of the parameters on the model. It is based on the principle of calculating the partial derivatives of the output functions in relation to the input

Building a Cormas model from scratch step by step: the ECEC model

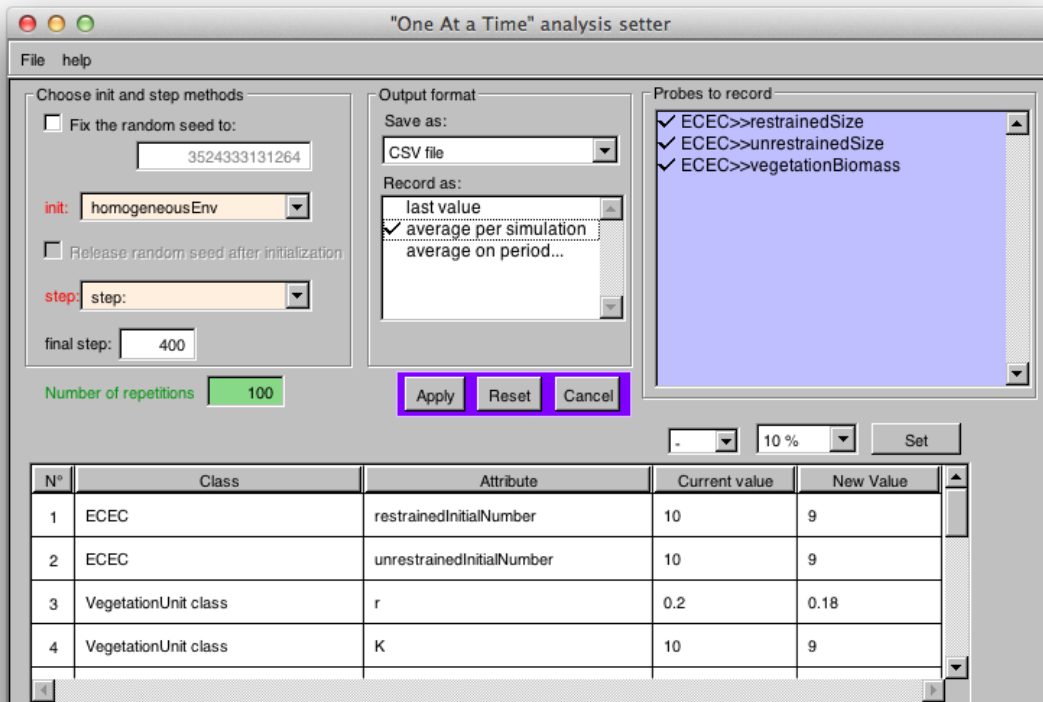
variables. More formally, we express the sensitivity $S_{i,j,t}$ of a parameter p_i as the partial derivative of the probe j at time t , relatively to p_i :

$$S_{i,j,t} = \frac{d\text{Probe}_{j,t}}{\delta \text{Param}_i}$$

This formula of sensitivity (also called local sensitivity) can only be applied to continuous and differentiable parameters and output values. For ABM, numerical approximations are necessary. It consists in varying the model inputs on a narrow range around a reference value.

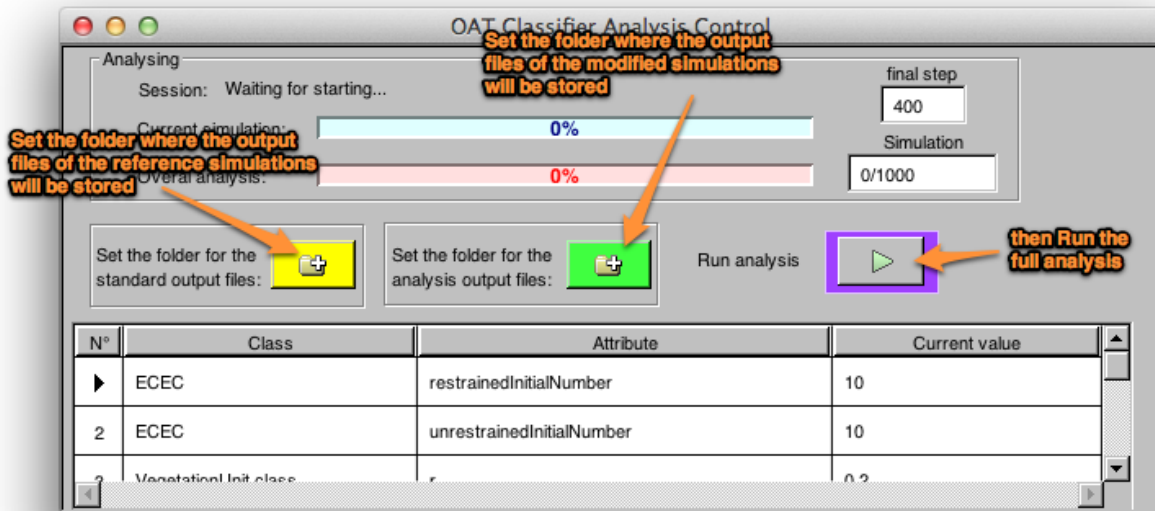
$$S(\text{Probe}_{j/\text{Param}_i}) = \left| \frac{\ln \left| \overline{\text{Probe}_{j/\text{Param}_i^{\text{modif}}} \right| - \ln \left| \overline{\text{Probe}_{j/\text{Param}_i^{\text{stand}}} \right|}{\ln \left| \text{Param}_i^{\text{modif}} \right| - \ln \left| \text{Param}_i^{\text{stand}} \right|} \right|$$

Cormas proposes a new tool to facilitate this analysis. On menu, select *Simulation* → *Analysis* → *OAT Sensitivity Analysis* → *Classify Parameters*. In the interface, similar to the previous one, select the *init* and *control* methods (#homogeneousEnv and #step:). Enter the final step (400), the number of repetitions (100), select the output format, the format of the probes (for example, ‘average on period...’ [150 to 400]) and the probes (*restrainedSize*, *unrestrainedSize* and *vegetationBiomass*). Then enter the new value for each parameter. It must be close to the reference value, around 10%. By clicking on ‘Set’ button, Cormas calculates these new values. It uses the default value to determine if a given parameter must be a float or an integer. For instance, -15% applied to *restrainedInitialNumber* won’t give 9.5 but a rounded value (9). A similar calculus is done for *K* (10) and *catabolicRate* (2). So, before Apply, check the list and change the necessary parameter’s new values (for instance, change 2 to 1.8 for *catabolicRate* as this parameter can be a float).



Building a Cormas model from scratch step by step: the ECEC model

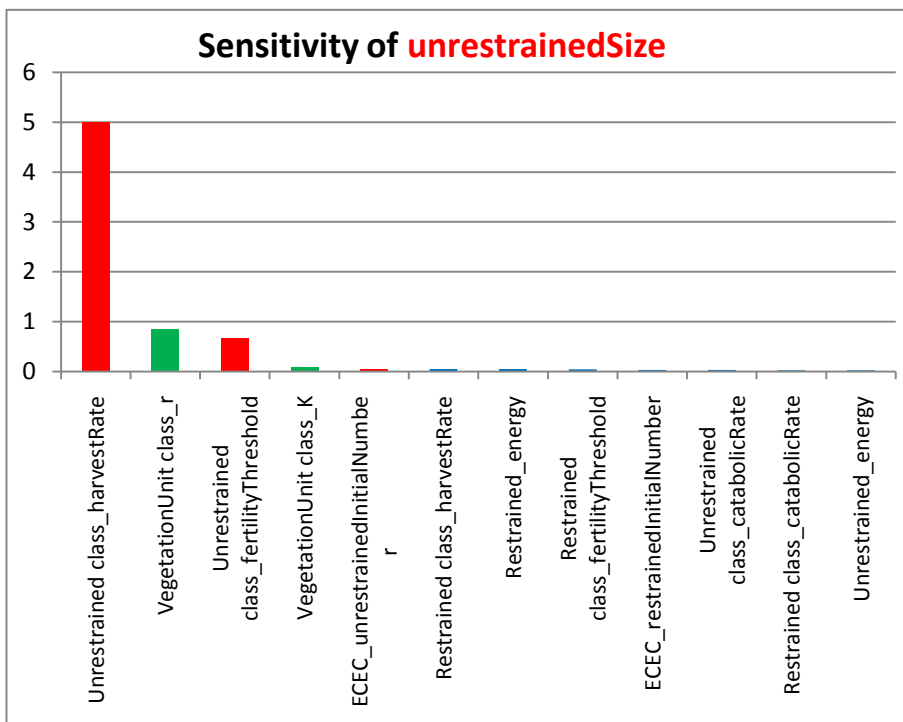
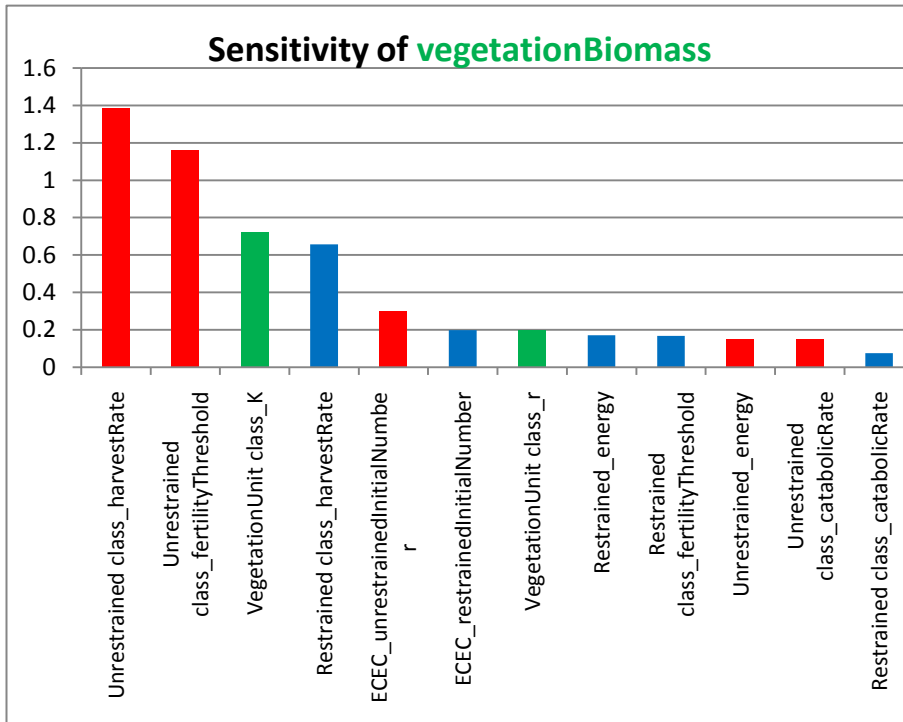
As explained by a pop-up message, the analysis will be performed in 2 sessions. A first one (100 simulations) to store the output files of the reference simulations. Then (automatically), a second one to run the simulations for the modified parameters.

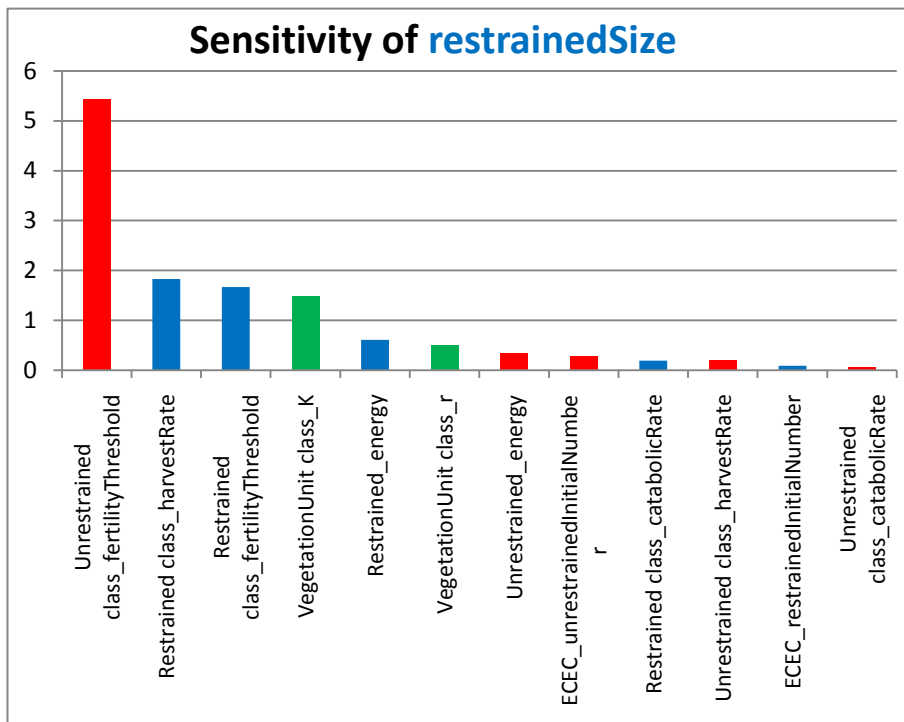


If the reference files are already available (due to a previous analysis), you can reuse them. For that, set the “folder for the output files” to the target folder.

When the complete analysis is done (may take a moment!), you should get these kinds of results (in Excel):

Building a Cormas model from scratch step by step: the ECEC model





By using the distance formula: $S_{i,tot} = \sqrt{\sum_j S_{i,j}^2}$, you can order the global sensibility of the parameters, as following:

