Structure of a model/program

Program myFirstModel
 	
global
Defines all global variables, model initialization and global behaviors.
 	
species mySpecies1
Defines variables, behaviors and aspects of agents of the species.
 	
experiment expName
Defines the way the model will be executed
Includes the type of the execution, which global parameters can be modified, and what will be displayed during simulation
	`model myFirstModel`
	
	`global {`
	`  // global variables declaration`
	`  // initialization of the model`
	`  // global behaviors`
	`}`
	
	`species mySpecies1 {`
	`  // attributes, initialization, behaviors and aspects of a species`
	`}`
	
	`experiment expName {`
	`  // Defines the way the model is executed, the parameters and the outputs.`
	`}`

Comments

Block comments	`/* A block comment starts with the an opening symbol /* .`
	`  The comment runs until the closing symbol below.`
	`*/`

Inline comments	`// This is an inline comment.` `// The // symbol have to be repeated before every line.`

Primitive types

Integer number
# value between -2147483648 and 2147483647 	
Real number
# absolute value between $4.9*10^{-324}$ and $1.8*10^{308}$ 	
String
# explicit value: "double quotes" or 'simples quotes' 	
Boolean value
*# 2 values: **true, false*** 	
	`int`
	
	`float`
	
	`string`
	
	`bool`

Other types

pair #with the two elements of undefined types
pair #with two elements of types type1 and type2
#explicit value using :: symbol: e.g. 1::"one"
color
#explicit value: rgb(255,0,0) for red. (3 components: Red, Green, Blue)
point
#explicit value: {1.0, 3} or {1.0, 3, 6}.
#Internal representation with 3 coordinates.
	`pair`
	`pair<type1, type2>`
	
	`rgb`
	
	`point`

Variable or constant declaration, affectation

Declaration of a global variable or an attribute
# Global variables and species attributes can be declared with or without initial value. 	
 	
# Local variables need to be initialized when they are declared. 	
# explicit declaration of the type 	
# (if the type of the affected value is different, this value is automatically casted to the declared type) 	
 	
Declaration of a global variable or an attribute with a dynamic value
# value computed at each simulation step 	
 	
# value computed each time the variable is used. 	
	`// Global variables or species attributes`
	`int an_int;`
	`string a_string <- "my string";`
	
	`// Local variables`
	`float a_float <- 10.0;`
	
	`// Global variables or species attributes with dynamic value`
	`// inc_int is incremented by 1 at each simulation step`
	`int inc_int <- 0 update: inc_int + 1;`
	
	`// random_int has a new random value each time it is used:`
	`int random_int -> { rnd(100) };`

Definition of a constant	`float pi <- 3.14 const: true;`
Affectation of a value to a variable Variable ← value or computed expression	`// Affectation of a value to an existing variable` `an_int <- 0;`

Display variables

Display ("Text: ", Expression)	`// Expression will be implicitly casted to a string`
	`// the + symbol is the string concatenation operator`
	`write "Text: " + Expression ;`

Conditionals

If Condition1 **then**
└ actions
	`if (expressionBoolean = true) {`
	`  // block of statements`
	`}`

If Condition1 **then**
└ action1
Else
└ other actions
	`if (expressionBoolean = true) {`
	`  // block 1 of statements`
	`} else {`
	`  // block 2 of statements`
	`}`

If Condition1 **then**
└ action1
Else If Condition2 **then**
└ action2
Else
└ other actions
 	
# composition of Boolean expressions 	
	`if (expressionBoolean = true) {`
	`  // block 1 of statements`
	`} else if (expressionBoolean2 != false) {`
	`  // block 2 of statements`
	`} else {`
	`  // block 3 of statements`
	`}`
	`// equal: = ; not equal: !=  (e.g. (var1 != 3)  )`
	`// Comparison: <, <=, >, >=  (e.g. (var2 >= 5.0)  )`
	`// logic operators : not (or !), and,  or  (e.g. (cond1 and not(cond2))  )`

Conditional affectation *# affectation depending of the condition value (if true, affects the value before the : symbol)*	`string s <- (expressionBoolean = true) ? "expression is true" : "expression is false";`

Loops

Repeat n times
└ actions
	`loop times: 10 {`
	`  write "loop times";`
	`}`

For index **from** 0 **to** n **Do**
└ actions
 	
# the index does not need to be declared before this loop 	
	`loop i from: 1 to: 10 step: 1 {`
	`  write "loop for " + i;`
	`}`

While Condition **Repeat**
└ actions
	`int j <- 1;`
	`loop while: (j <= 10) {`
	`  write "loop while " + j;`
	`  j <- j + 1;`
	`}`

For each element **of** a container **Do**
└ actions
 	
# the variable containing each element does not need to be declared before this loop 	
	`list<int> list_int <- [1,2,3,4,5,6,7,8,9,10];`
	`loop i over: list_int {`
	`  write "loop over " + i;`
	`}`

For each agent of a species or a set of agents **Do**
└ actions executed in the context of the agent
 	
*# in the ask, **self** keyword refers to the current agent (i.e. each agent of the species parameter of the ask) and **myself** refers to the agent calling the ask statement.* 	
	`ask mySpecies2 {`
	`  // statements`
	`}`
	`ask list_agent {`
	`  // statements`
	`}`

Declaration of a procedure / an action

Procedure ProcedureName
└ actions
	`action myAction {`
	`  write "Action without param";`
	`}`

Procedure ProcedureName (pd1, pd2)
└ actions
	`action myActionWithParam( int int_param,`
	`  string my_string <- "default value") {`
	`  write my_string + int_param;`
	`}`

Call of a procedure / an action

Call ProcedureName
Call ProcedureName (pa1, pa2, pa3)
 # if a parameter has a default value, it can be omitted when calling the action. It will thus have the default value.

 # if the procedure has been defined in another species, the current agent has to ask an agent of this species to call the procedure.

```
do myAction;          // equivalent to do myAction();
do myActionWithParam(3, "other string");

do myActionWithParam(3);  // the second parameter has its default value

ask an_agent {
   do proc(3);
}
```

Declaration of a function

Function FunctionName **: type**
  actions
  return value

Function FunctionName (pd1, pd2) **: type**
  actions
  return value

```
int myFunction {
   return 1+1;
}


int myFunctionWithParam(int i, int j <- 0){
   return i + j;
}
```

Call of a function

Variable ← FunctionName ()

Variable ← FunctionName (pa1, pa2)
 # if a parameter has a default value, it can be omitted when calling the action. It will thus have the default value.
 # if the function has been defined in another species, the current agent has to ask an agent of this species to call the function.

```
// the current agent calls the function
int i <- myFunction();
int j <- self myFunction();

// The current agent calls a function with parameters
int l <- myFunctionWithParam(1);
int m <- myFunctionWithParam(1,5);


// another agent calls a function with parameters
int n <- an_agent myFunctionWithParam(1,5);
```

List, map and matrix

Declaration and explicit initialization of list, map and matrix variables.

```
list<int> list_int <- [1,2,3,4,5];
map<int,string> map_int <- map([1::"one",2::"two"]);
matrix<int> m <- matrix([[1,2],[3,4]]);
```

Incremental creation of lists and maps

 # Replacement of an element from list or matrix.
 # In map, we can replace the value associated to a key.

```
// Add 7 at the end of the list
add 7 to: list_int;
// Add the pair 6::"six" to the map
add "six" at: 6 to: map_int;

put 8 at: 5 in: list_int;
put 7 at: {0,0} in: m;
```

Access to elements
 # List access using the index, map access using the key, matrix access using coordinates in the matrix.
 # the first element of a list has an index of 0.

```
// Access of an list element out of bounds will throw an error, Access to the value associated to a non-existing key will return nil
list_int[1]
map_int[2]
m[{1,1}]
```

Loop over elements of a list, map, matrix
 # Loop over maps have to be done on keys, values or pairs list

```
// loop over values of a list
loop i over: list_int {   }

// loop over values of the map (similar with keys and pairs)
loop i over: map_int.values {     }
```

Use of an external model

Use a model (i.e. its species and global variables and behaviors) defined in another file.

```
// this should be after the model statement
import "otherModels/model2.gaml"
```

Definition of a species

Species SpeciesName
  Definition of the set of attributes

```
species mySpecies1 {
   int s1_int;
   float energy <- 10.0;
```

init
statements
 	
behavior behaviorName
statements
 	
aspect aspectName
statements to draw the agents
 	
# built-in attributes: name, shape, location… 	
	`init {`
	`  // statements dedicated to the initialization of agents`
	`}`
	
	`reflex reflex_name {`
	`  // set of statements`
	`}`
	
	`aspect square {`
	`  draw square(10);`
	`  draw circle(5) color: #red ;`
	`}`
	`}`

Use of an architecture
# by default, species use the reflex architecture 	
# Agents can still use reflex behaviors, even with another architecture. 	
	`species mySpeciesArchi control: fsm {`
	
	`}`

Use of skills
# by default, no skill is associated with a species. 	
# A skill provides additional attributes and actions. 	
	`species mySpecies3 skills: [moving, communicating] {`
	
	`}`

Inheritance *# No multiple inheritance is allowed.*	`// mySpecies2 gets all attributes and behaviors from mySpecies1` `species mySpecies2 parent: mySpecies1 {  }`

Creation of agents

Creation of N agents of a species
# Agent creation is often done in the global init. 	
Creation of N agents of a species
Initialization of the agents
	`create mySpecies1 number: 10;`
	
	`create mySpecies1 number: 20 {`
	`  an_int <- 0;`
	`}`

Creation from (shapefile) data *# Objects of the file have an id attribute.*	`create mySpecies1 from: a_shp_file` `  with: [an_int::int(read('id'))];`

Definition of an experiment

experiment expName **type: gui**
Set of parameters
 	
Outputs definition
display
species, grid, agents
display
chart
data
 	
#As many displays as needed can be created (charts or agent display). Each represents a point of view on the simulation.
experiment expName **type: batch**
Set of parameters
Exploration method
 	
Outputs definition
display
chart
data
 	
#In the batch experiment, charts can be used to plot the evolution over the simulations of a global indicator.
	`experiment expeName type: gui {`
	`  parameter "A variable" var: an_int <- 2`
	`  min: 0 max: 1000 step: 1 category: "Parameters";`
	`  output {`
	`  display display_name {`
	`  species mySpecies2 aspect: square;`
	`  species mySpecies1;`
	`  }`
	`  display other_display_name {`
	`  chart "chart_name" type: series {`
	`  data "time series" value: a_float;`
	`  }`
	`  }`
	`  }`
	`}`
	`// repeat defines the number of replications for the same parameter values`
	`// keep_seed means whether the same random generator seed is used at the first replication for each parameter values`
	`experiment expeNameBatch type: batch repeat: 2`
	`  keep_seed: true until: (booleanExpression) {`
	`  parameter "A variable" var: an_int <- 2 min: 0 max: 1000 step: 1 ;`
	`  method exhaustive maximize: an_indicator ;`
	
	`  permanent {`
	`  display other_display_name {`
	`  chart "chart_name" type: series {`
	`  data "time series" value: a_float;`
	`  }`
	`  }`
	`  }`
	`}`