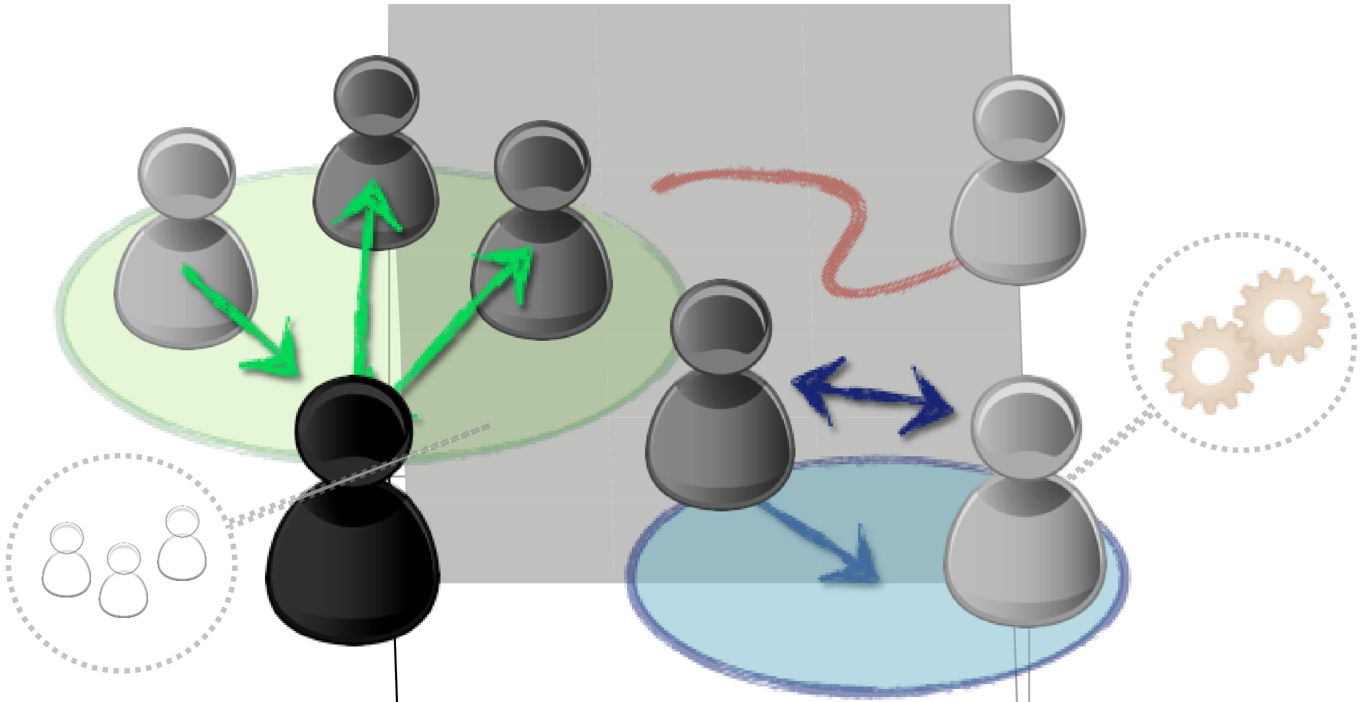


Cours 1c: Introduction a Gama

tiré de la GAMA Winter School Nov. 2012, Can Tho University / IRD

Alexis Drogoul (a, d), Benoit Gaudou (b), Patrick Taillandier (c)
Philippe Caillou (e), Arnaud Grignard (a), Nicolas Marilleau (a), An Duc Vo (a)

(a) UMI 209 UMMISCO, IRD / UPMC
(b) UMR 5505 IRIT, CNRS / Université de Toulouse
(c) UMR 6266 IDEES, CNRS / Université de Rouen
(d) JEAI DREAM, IRD / Université de Can Tho
(e) LRI, INRIA TAO Project / Université d'Orsay

General introduction to GAMA

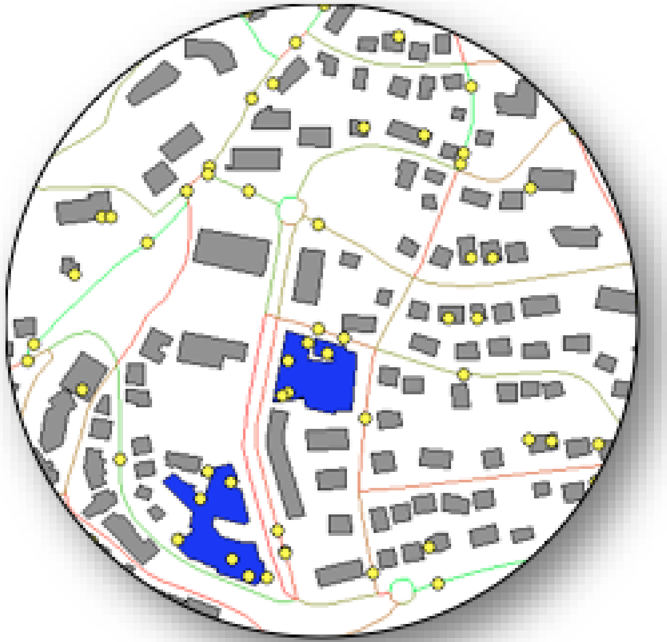
- Software platform dedicated to building spatially explicit agent-based simulations

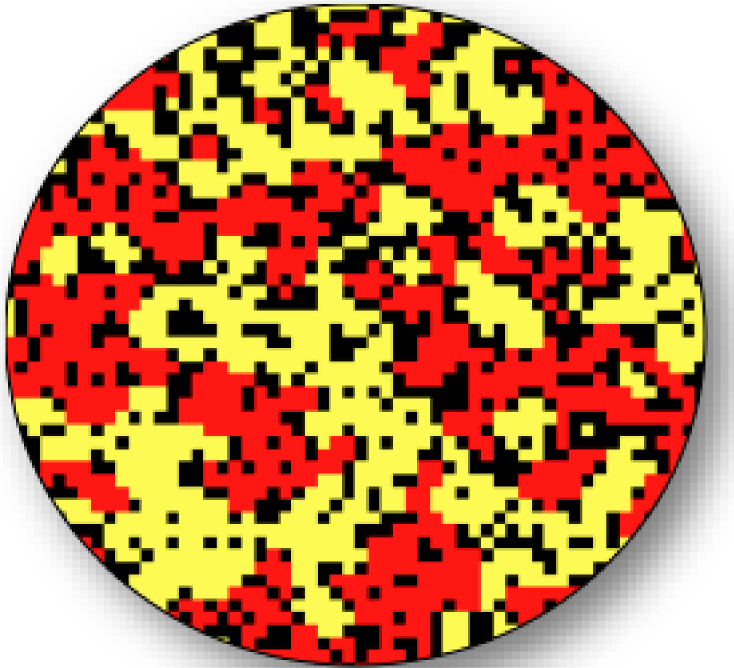
- **Generic : can be used for a wide range of applications**
- **Developed under GPL/LGPL license : free**
- **Integrates a complete modeling language (GAML) and an integrated development environment: allows modelers (even non computer-scientists) to build models quickly and easily**
- **Developed in JAVA : easy to extend in order to take specific needs into account**
- **Integrates tools to analyze models: parameters space exploration and calibration of models**

- Strengths of GAMA vs other Simulation Frameworks (Netlogo, Repast, Cormas, ...)

- **Supports the development of quite complex models**
- **Seamless integration of geographic data and GIS tools with agent-based models**
- **Integrates a methodological approach to define multi-level models**
- **Integrates high-level tools: multi-criteria decision making tools, clustering functions, statistical operators…**
- **Easily extensible thanks to its open architecture, which relies on two legacy Java technologies : OSGI plugin framework and Java annotations**

- Some examples





More information

- Blog
  http://gama-platform.blogspot.fr/

- Facebook
  http://www.facebook.com/GamaPlatform

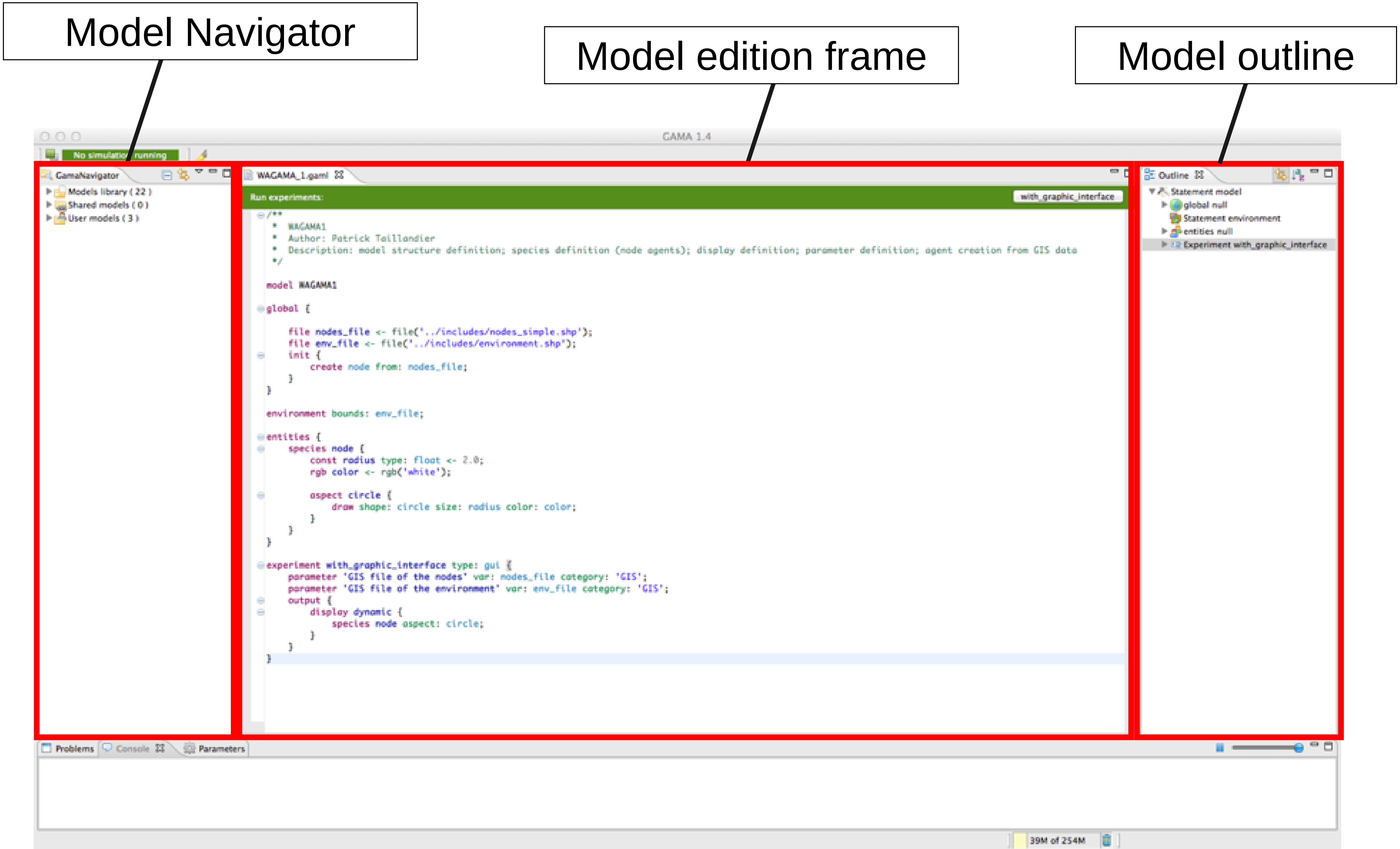
- Web site of the project
  http://code.google.com/p/gama-platform/

- Documentation
  http://code.google.com/p/gama-platform/wiki/Documentation
- Mailing-lists

- **General mailing-list**
  https://groups.google.com/forum/?fromgroups#!forum/gama-platform

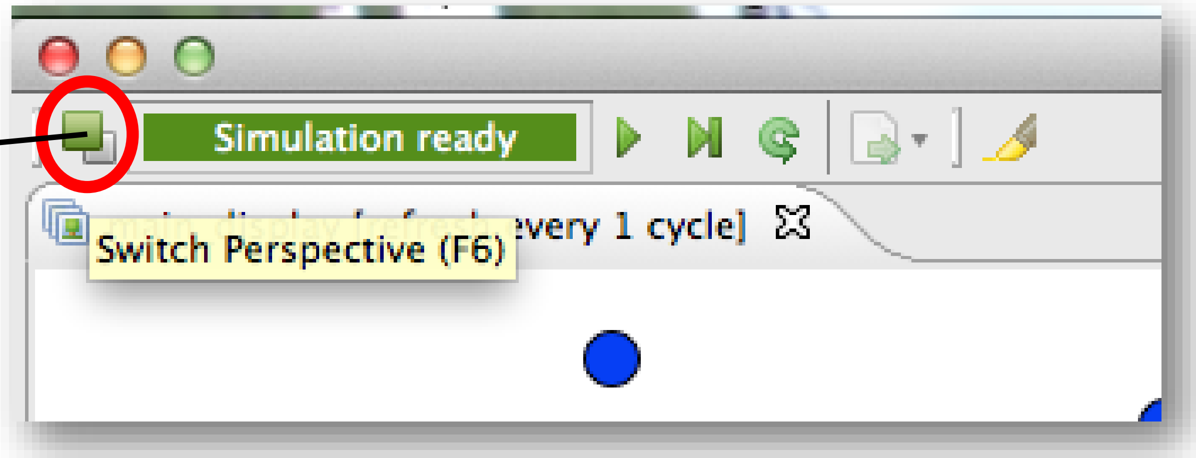
- **Developers mailing-list**
  https://groups.google.com/forum/?fromgroups#!forum/gama-dev

User interface of GAMA

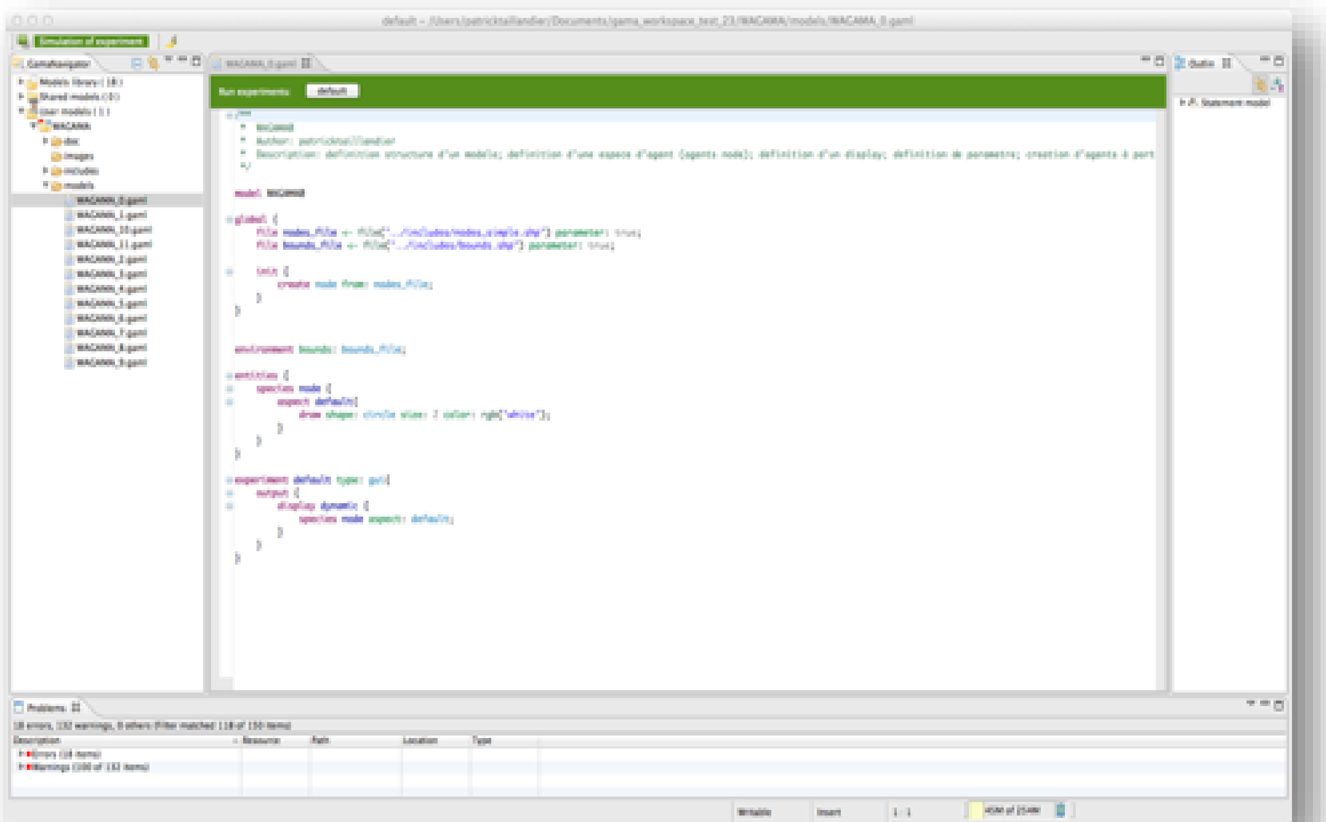


Modeling and simulation perspectives

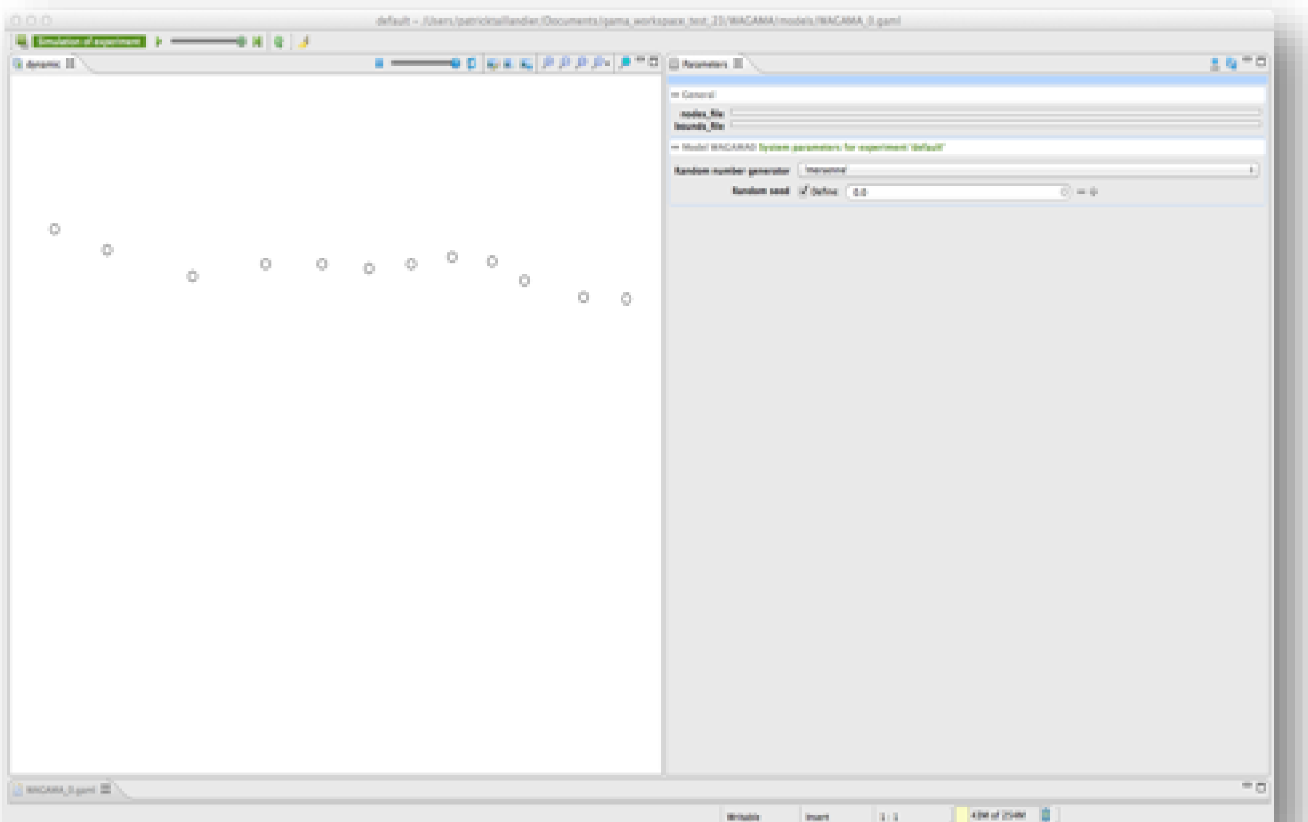
Perspective switch button



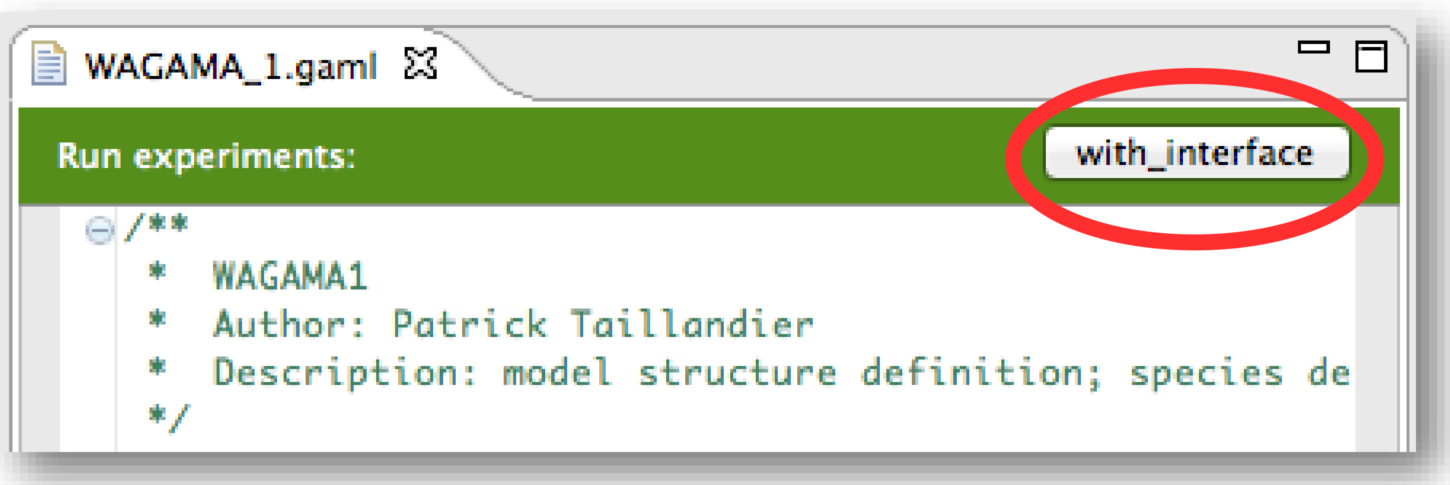
Modeling perspective



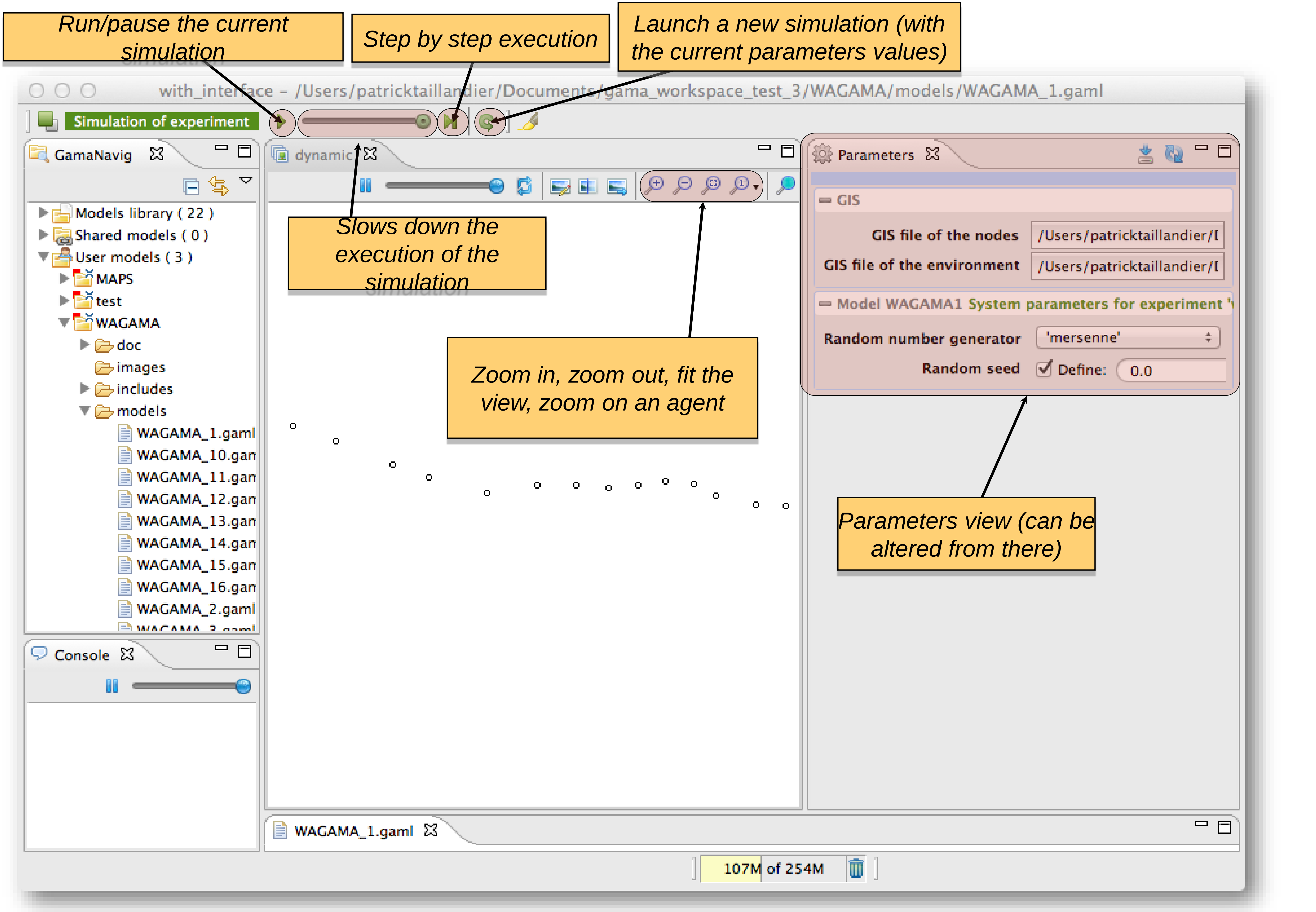
Simulation perspective



- In the modeling perspective, click on the desired experiment button (these buttons only appear when the experiments can be launched safely, i.e. the model does not contain any error)

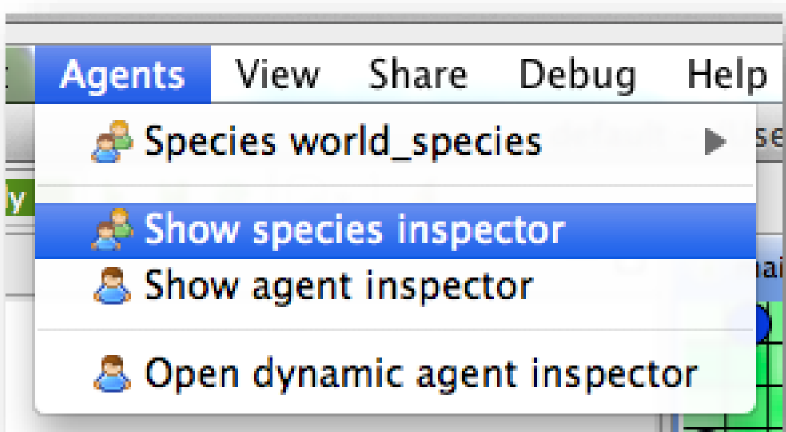


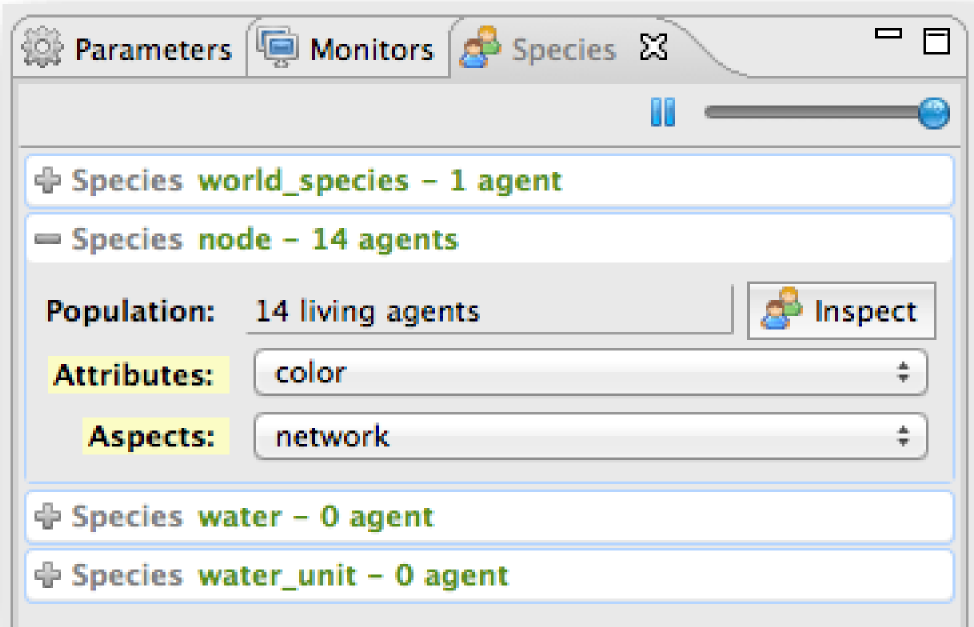
Simulation interface

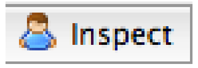


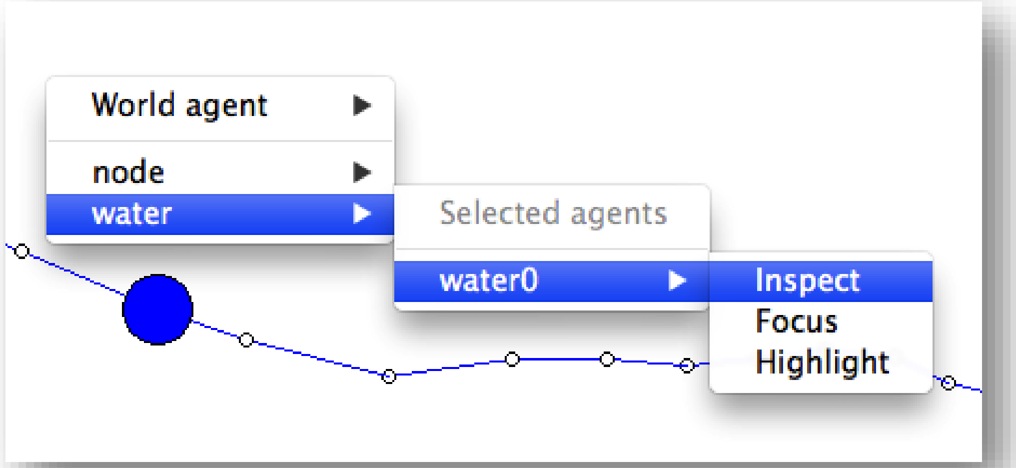
❖ **Inspector**: provides informations about a species or an agent

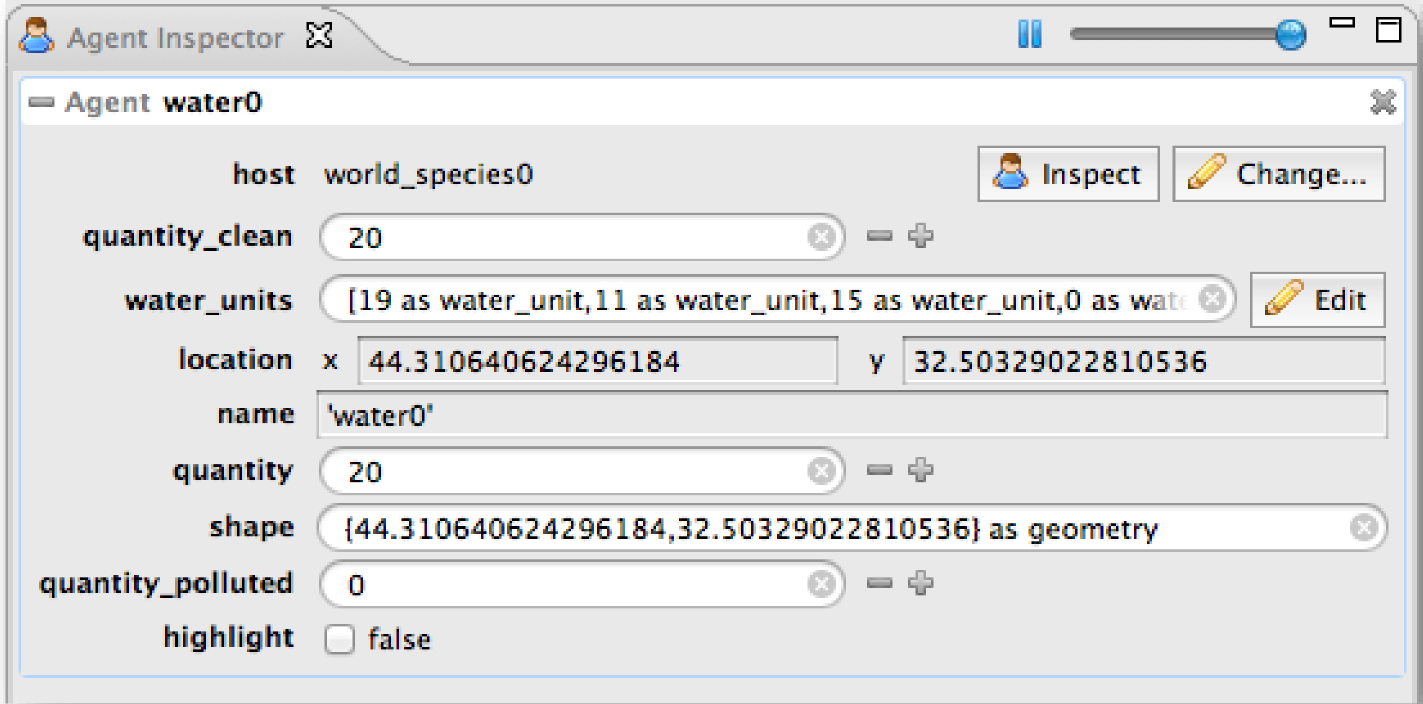
- ***Species inspector***: provides informations about all the species present in a model
  - Available in the *Agents* menu





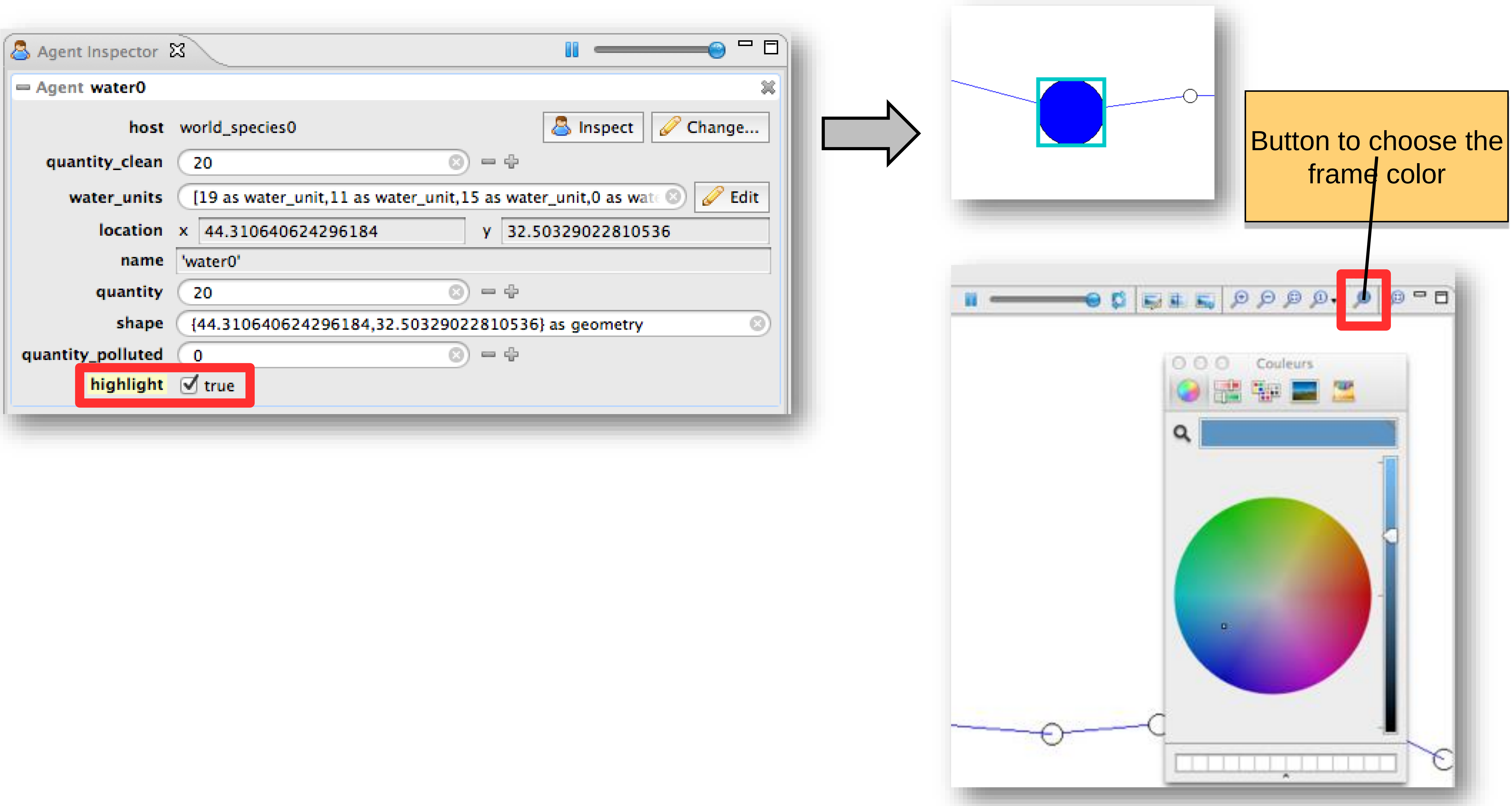
- **Inspector**: provides informations about a species or an agent
  - ***Agent inspector***: provides information about one specific agent. Also allows to change the values of its variables during the simulation.
    - Available from the *Agents* menu, by right_clicking on a display, in the species inspector or when inspecting another agent (button 





Inspectors - 3

- ***Agent inspector***: provides information about one specific agent. Also allows to change the values of its variables during the simulation.
  - Possibility to «highlight» the selected agent



The bases of GAMA through the Schelling model example

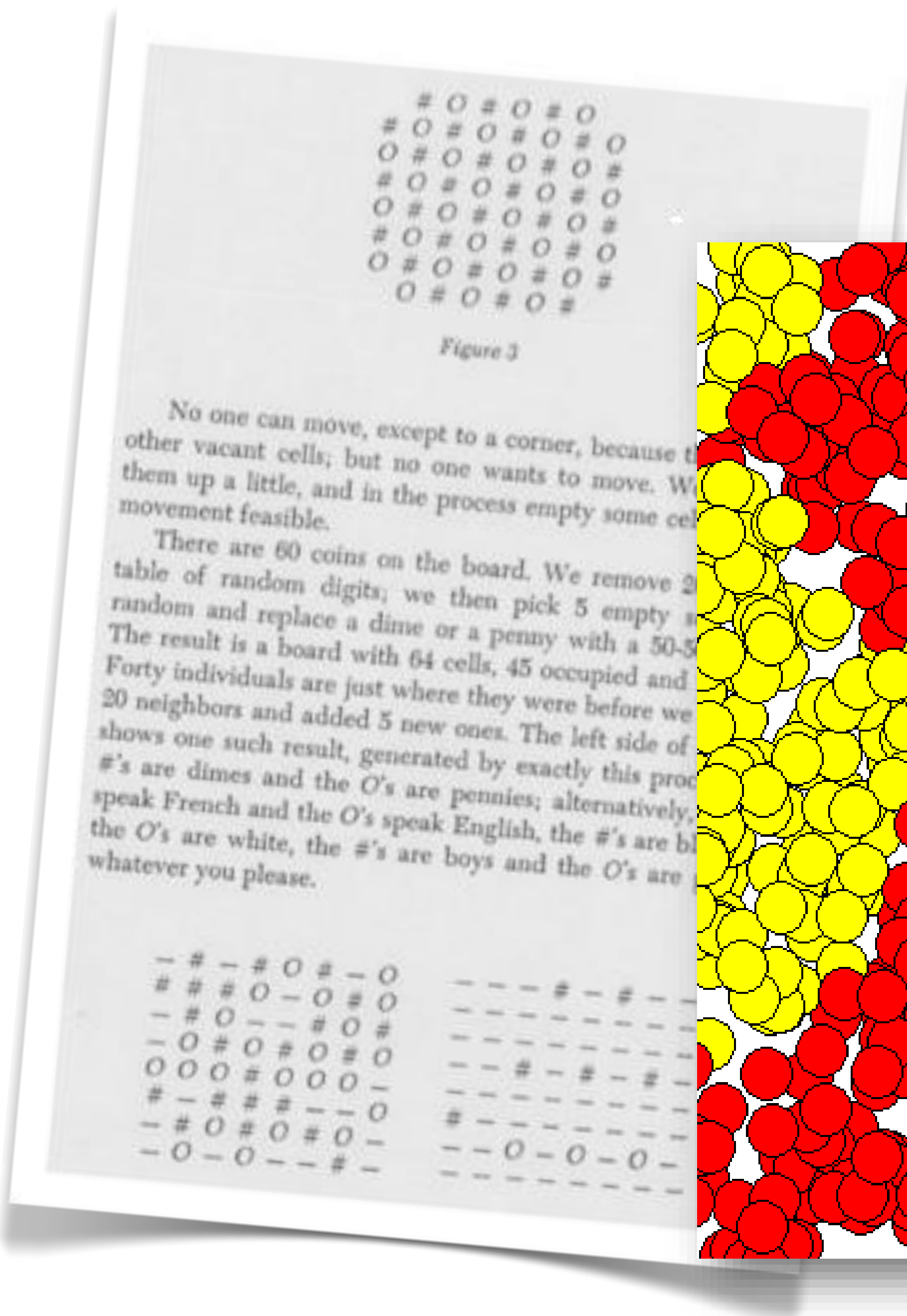
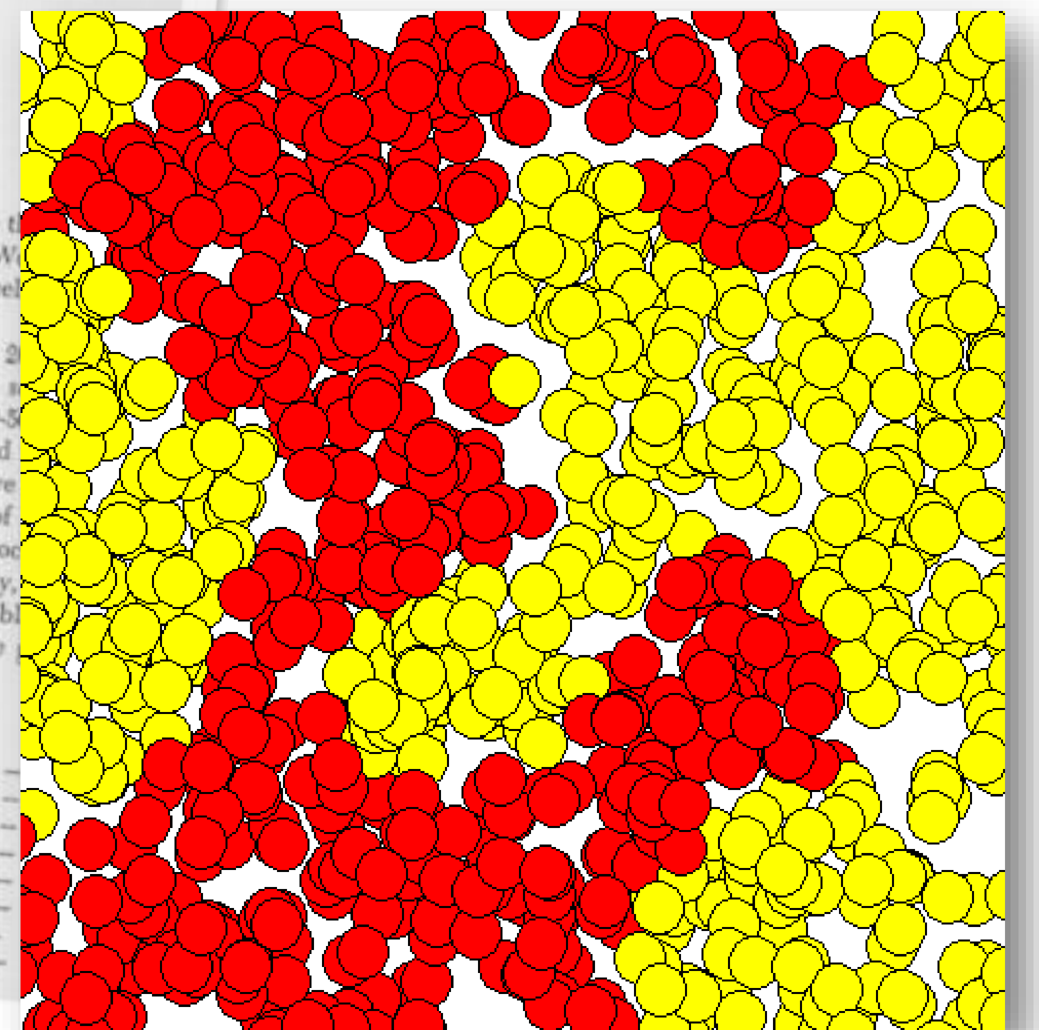


Figure 3

No one can move, except to a corner, because t
other vacant cells; but no one wants to move. W
them up a little, and in the process empty some ce
movement feasible.

There are 60 coins on the board. We remove 2
table of random digits; we then pick 5 empty
random and replace a dime or a penny with a 50-5
The result is a board with 64 cells, 45 occupied and
Forty individuals are just where they were before we
20 neighbors and added 5 new ones. The left side of
shows one such result, generated by exactly this proc
#'s are dimes and the O's are pennies; alternatively,
speak French and the O's speak English, the #'s are bl
the O's are white, the #'s are boys and the O's are
whatever you please.

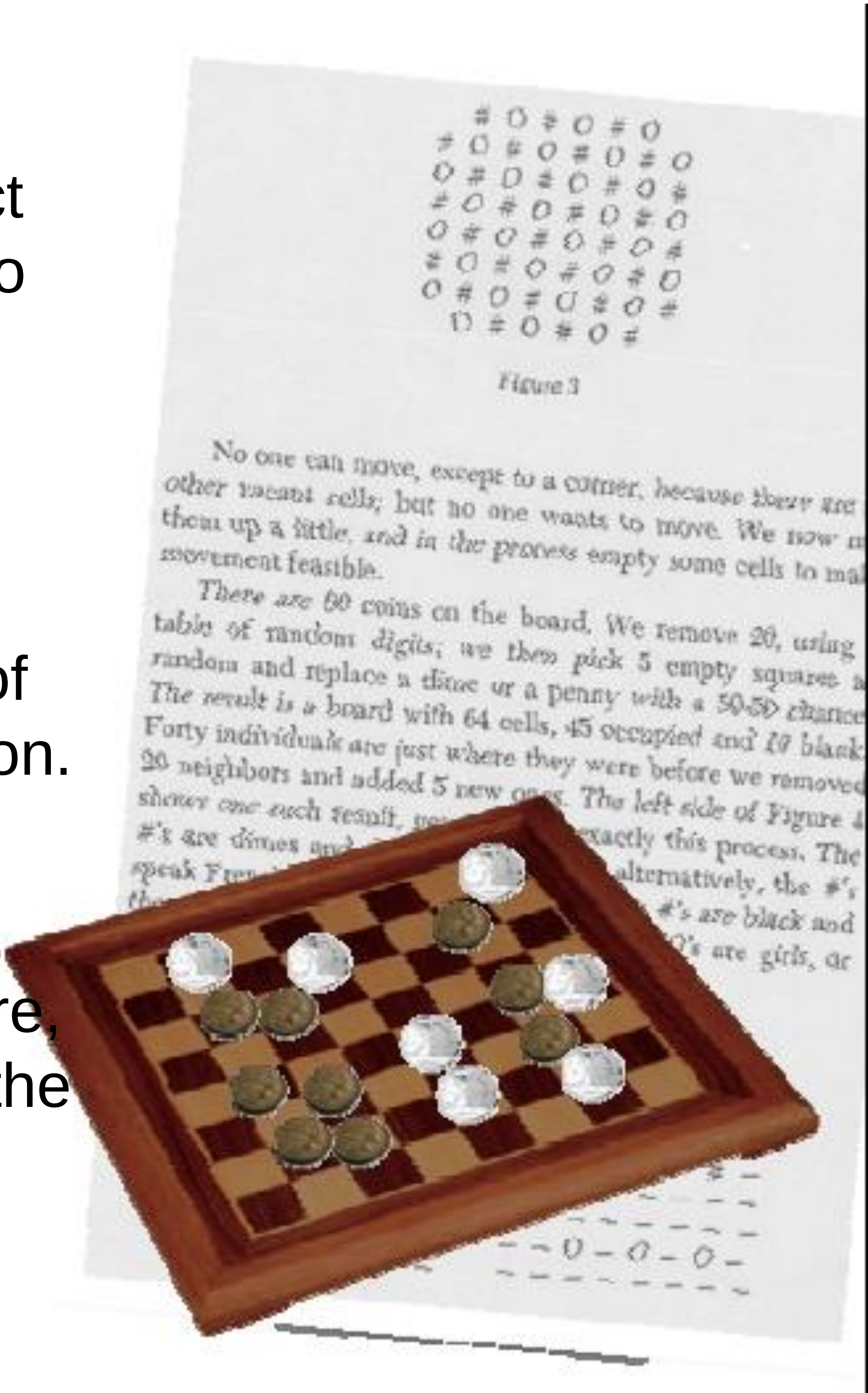


GAMA through an example: Introduction to Schelling's model

- In 1969, Schelling introduced a model of segregation in which individuals of two different colors, positioned on a grid abstract representation of a district), choose where to live based on a preferred percentage of neighbors of the same color.

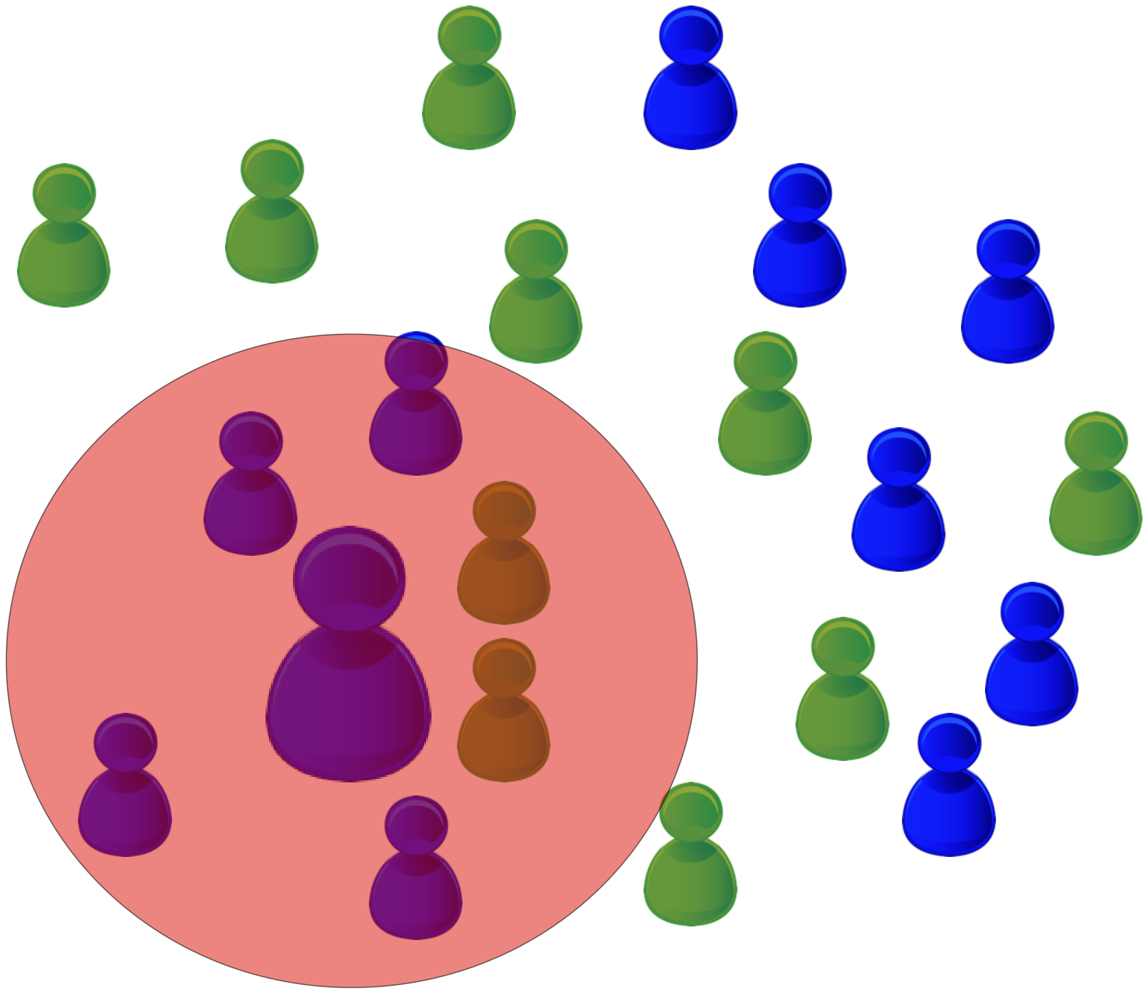
- Using coins on a board, he showed that a small preference for one's neighbors to be of the same color could lead to total segregation.

- It is a good example of a generative model, where the emergence of a phenomenon here, segregation) is not directly predictible from the knowledge of individual

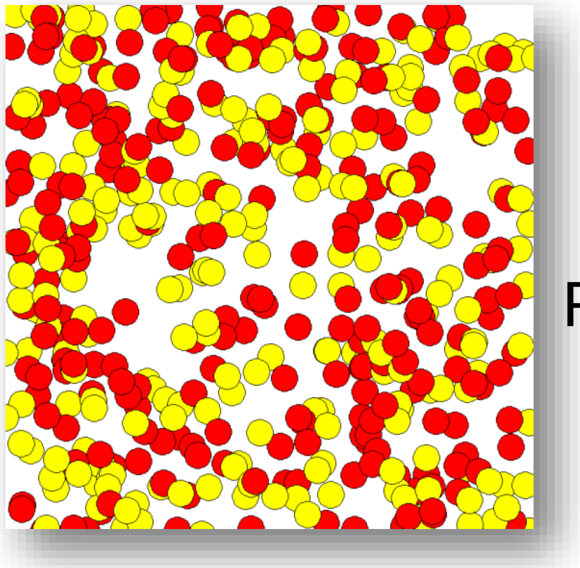


In the simplest agent based model, agents (people) are randomly placed in a continuous space. Each agent has a color, a perception of its neighbors and a preference - i.e. a minimal desired percentage of neighbors of the same color.

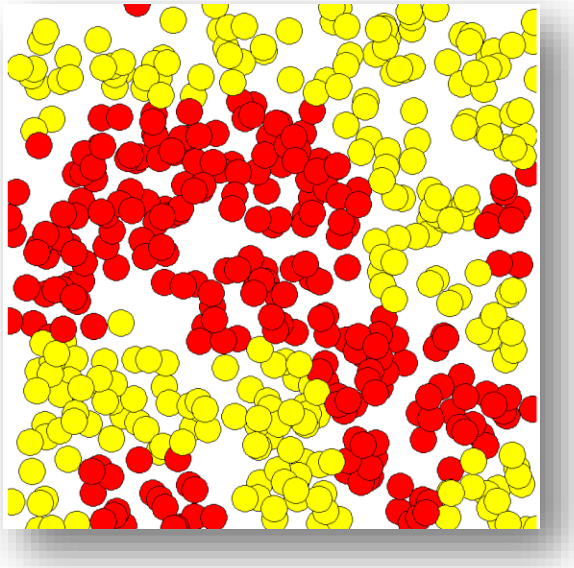
Only behavior of the agent: if the percentage of neighbors of the same color is inferior to the preference, then the agent randomly moves to another location of the space.



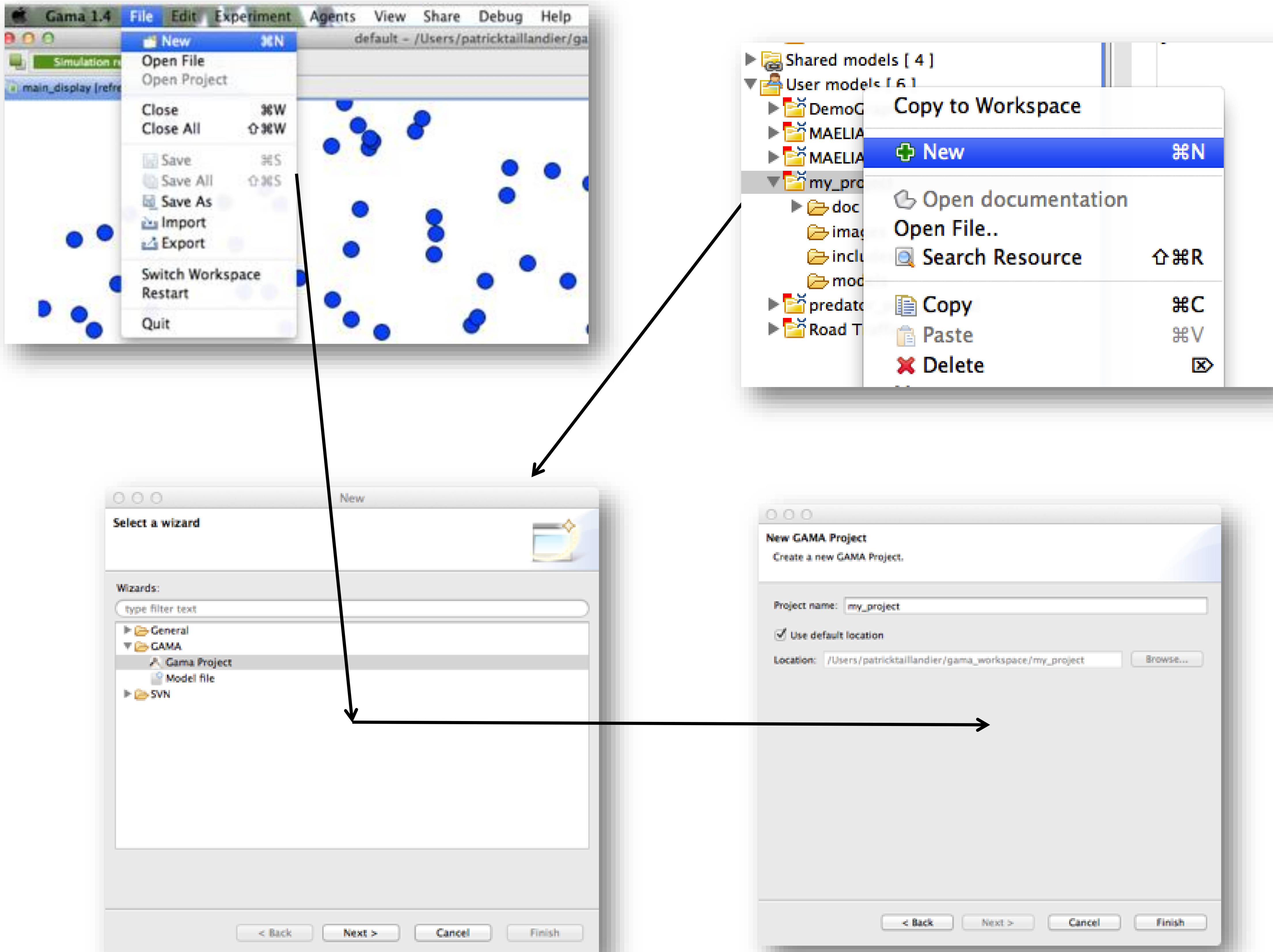
Neighborhood : 4 neighbors of the same color, and 2 neighbors of a different color



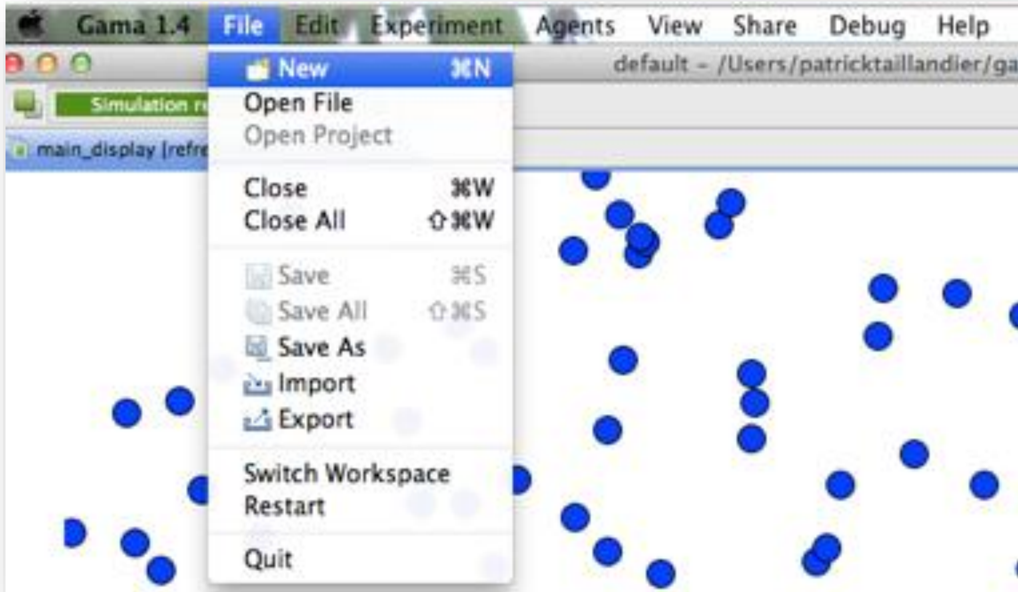
Result after 30 simulation steps

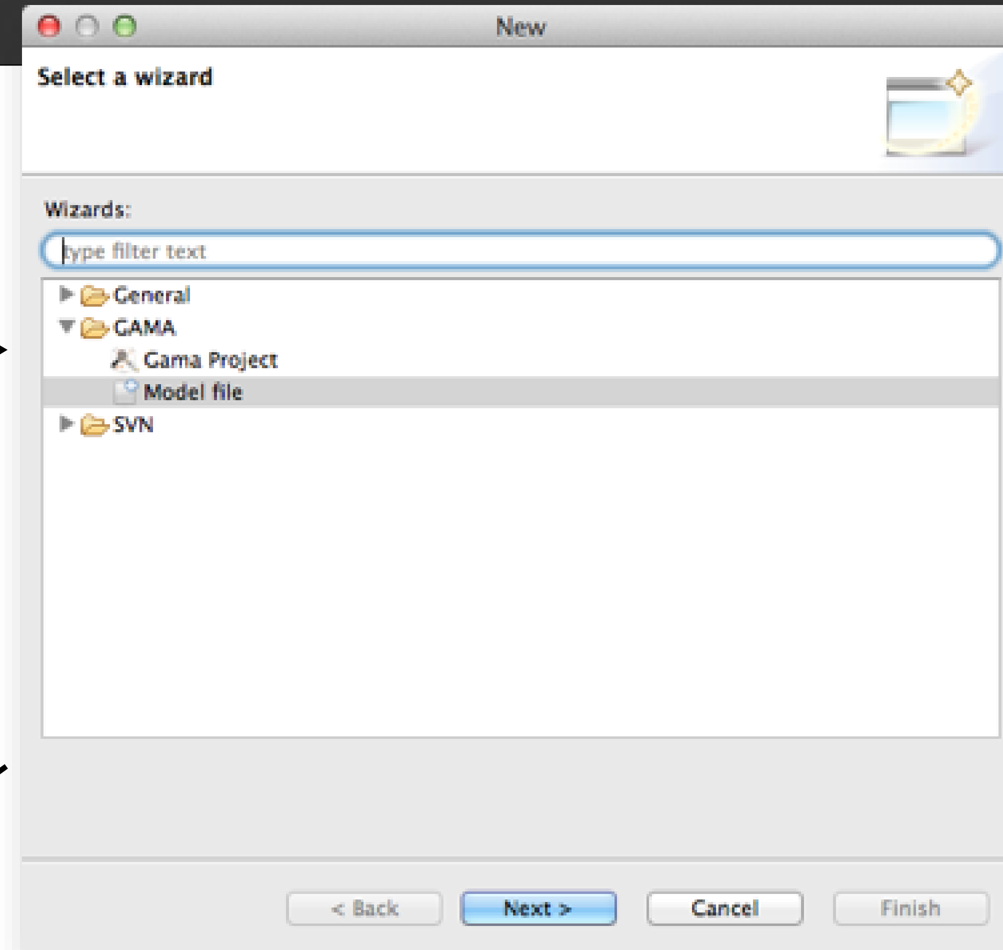


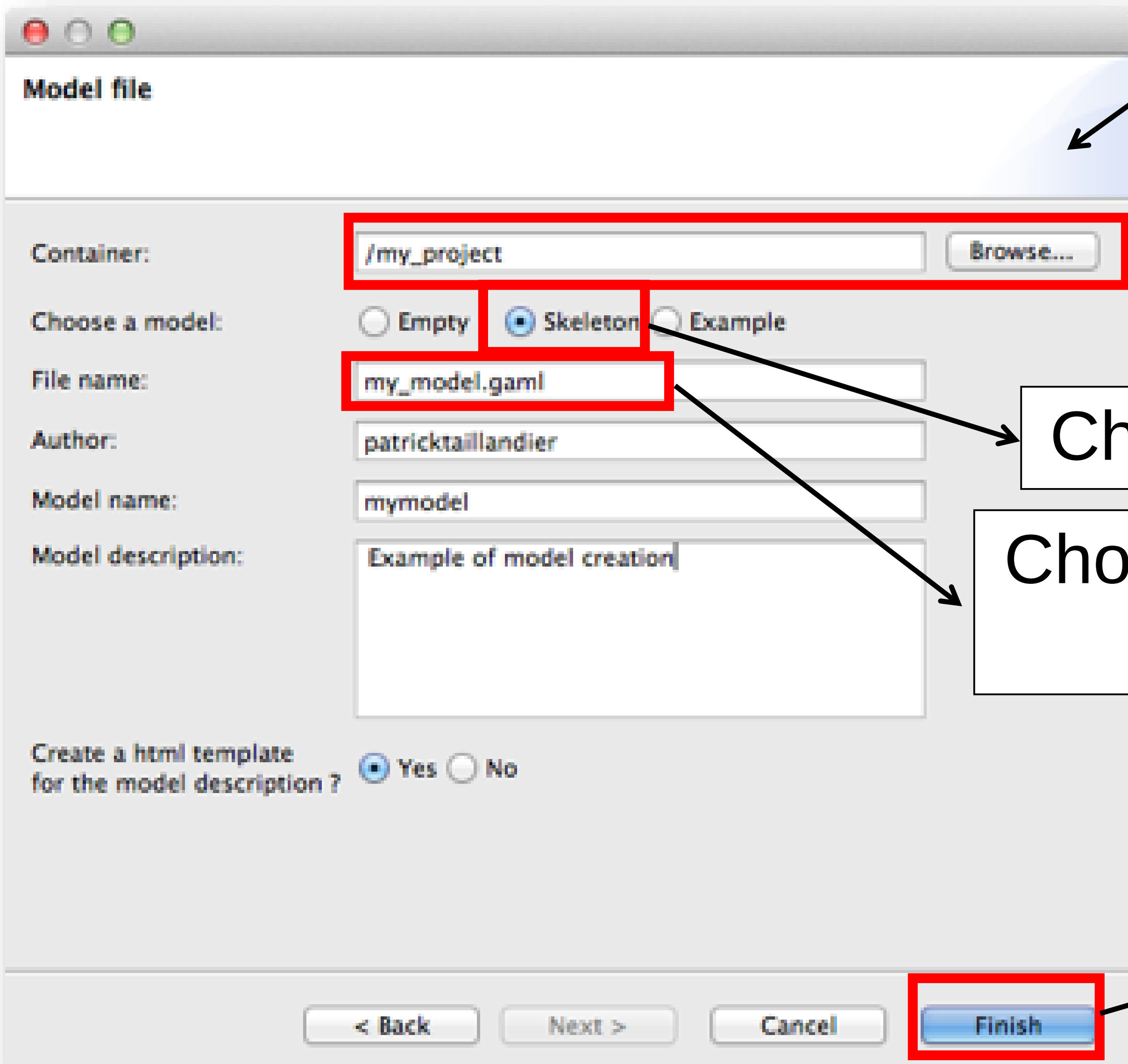
Creating a new project



Creating a new model







Choose the project

Choose skeleton

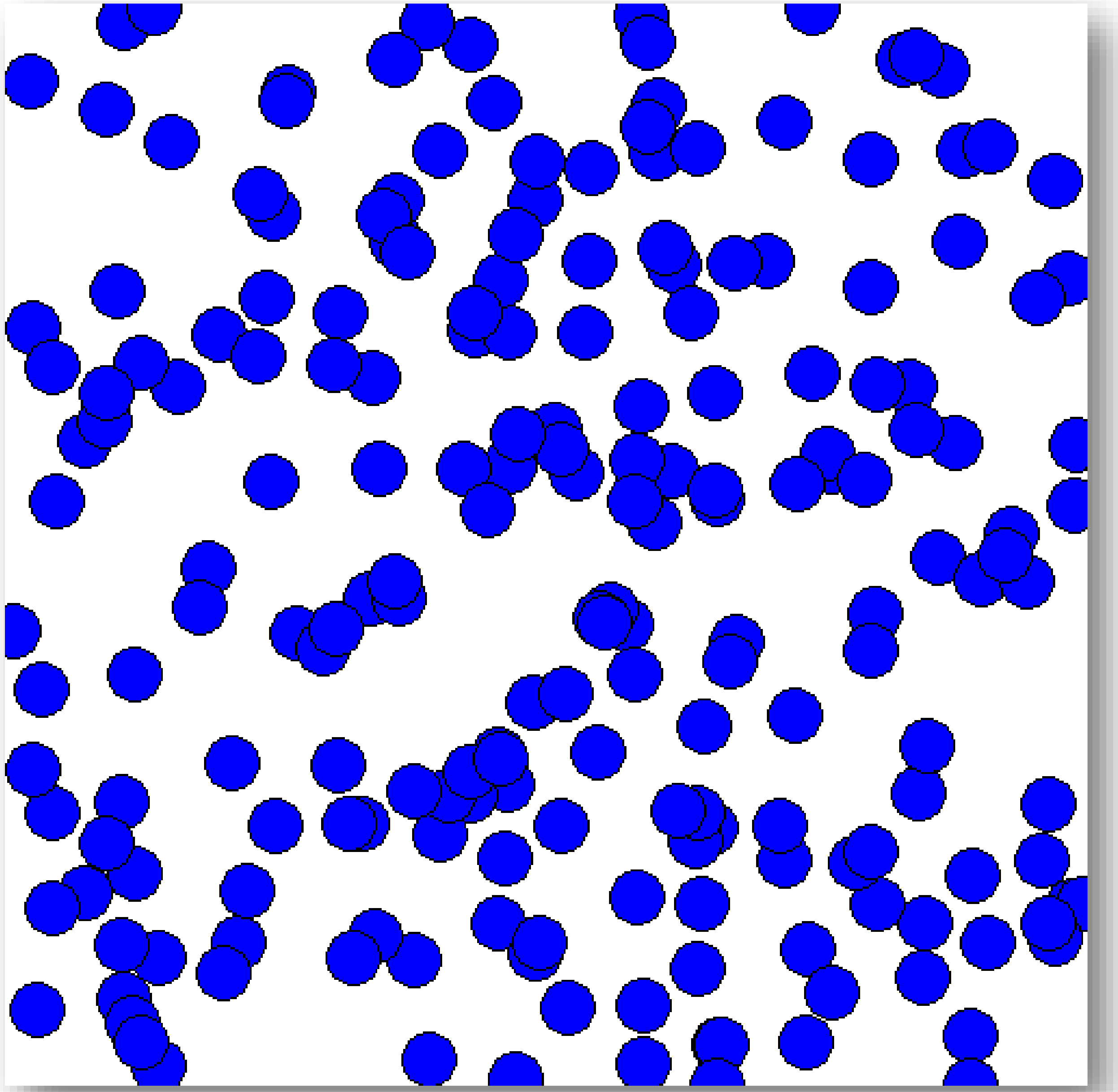
Choose the name of the file

Click on Finish

Step 1: basic model

❖ Objectives:

- Definition of the people species
- Creation of 500 people agents randomly located in the environment
- Display of the agents



❖ Key points:

- Introduction to the structures of GAMA models
- Definition of a species
- Creation of agents
- Display of agents

GAML

- **Complete modeling language**: simple structures, but very rich in terms of operators
- **Supported by an IDE** (Integrated Development Environment): ease the writing of models

Some basic rules :

- A statement always ends with an " ; " or with a block.
- A block is delimited by "{" and "}" and contains a sequence of statements.
- A statement is identified by a keyword, followed by a number of facets (its «parameters»)

GAMA model structure

❖ Four statements define the main sections:

- Global : variables, actions, dynamics and global initializations
- Environment : properties of the global environment
- Entities : species of agents
- Experiment : execution context of simulations, defining for instance their inputs and outputs. Several experiments can be defined in a same model.

```
model mon_model

global {
    /** Insert the global definitions,
     * variables and actions here
     */
}

environment {
    /** Insert the grids and the properties
     * of the environment
     */
}

entities {
    /** Insert here the definition of
     * the species of agents
     */
}

experiment default type: gui {
    /** Insert here the definition of the
     * default gui experiment
     */
}
```

Two ways of writing comments in your model :
//… : inlined comments. Example : //This is a comment (always on one line)
/* … */ : block comments. Example : /* This is a block comment (possibly on several lines) */

Step 1: Species definition

```
model CT_Schelling

global {
}

environment {
}

entities {
}

experiment schelling type: gui {
}
```

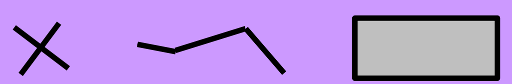
- A species represents a «prototype» of agents: defines their common properties
- A species includes several sub-definitions
- The internal state of its agents (variables)
- Their behavior
- How they are displayed (aspects)

For our model:

Name of the species

```
entities {
  species people {

  }
}
```

All the species inherit from predefined built-in variables:

- A name (name)
- A shape (shape) 
- A location (location) : the centroid of its shape.

Step 1: Species definition: aspect

- An *aspect* represents a possible way to display the agents of a species :

aspect** aspect_name* ***{…}

- In the block of an *aspect,* it is possible to draw : a predefined shape (circle…), the shape of the agent, an image or text…

```
model CT_schelling

global {
}

environment {
}

entities {
}

experiment schelling type: gui {
}
```

For our model:

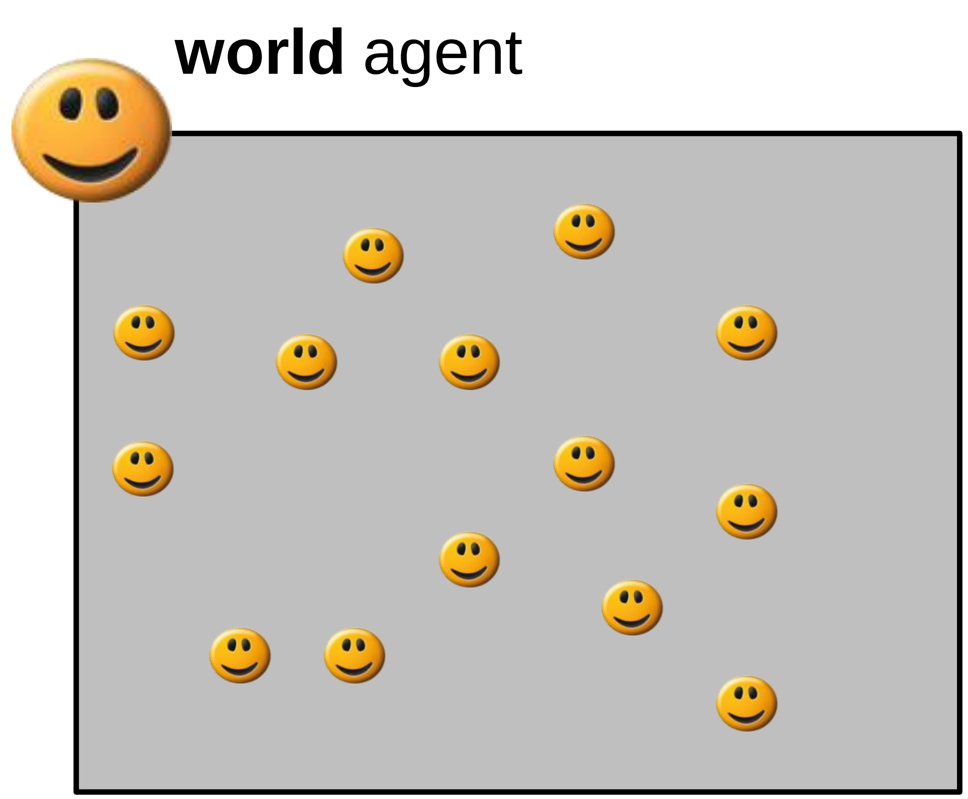
```
entities {
  species people {
    aspect base {
      draw circle(5) color: rgb("yellow");
    }
  }
}
```

The *base* aspect allows to display each people agent in the form of a yellow circle of radius 5 centered at the agent location

Step 1: Global block: definition of the initialization sequence

- Actually the definition of the species of a specific agent (called world)
- Represents everything that is global to the model : dynamics, variables…
- Allows to init simulations (init block) : the world is always created and initialized first when a simulation is launched.
- The geometry (shape) of the world is a rectangle defined in the environment section.

```
model CT_Schelling

global {
}

environment {
}

entities {
}

experiment schelling type: gui {
}
```



Step 1: Global block: definition of the initialization sequence

- Creation of agents : statement **create** species_name +
  - number : number of agents to create (int, 1 by default)
  - from : GIS file to use to create the agents (string or file)

  One of the two
  If nothing is specified, creation of an agent of the same species as the caller

  - returns: liste des agents créés (list)

For our model: creation of 500 agents of species *people*

Agent creation of the specified species

```
global {
  init {
    create people number: 500;
  }
}
```

By default, agents when created are located randomly in the environment (except when a GIS file is used or they explicitely initialize their *location*)

```
model CT_Schelling

global {
}

environment {
}

entities {
}

experiment schelling type: gui {
}
```

Step 1: Environment definition

- GAMA provides a *continuous environment* to each model *which* is the geometry of the *world* agent
- Definition of the size of the environment
  - Using the *width* and *height* facets
  - Using the *bounds* facet, with :
    - a *point* ({x,y}),
    - a *shapefile* (GIS) : envelope of all the data contained in the shapefile
    - a raster file (*asc)*
    - a list of files (union of their envelopes)
- By default, the environment is a rectangle of 100 x 100

For our model: use of the default environment

```
environment {}
```

```
model CT_Schelling

global {
}

environment {
}

entities {
}

experiment schelling type: gui {
}
```

Step 1: Experiment

- An experiment block defines how a model can be simulated (executed).
- Several experiments can be defined for a model.
- They are defined using:

  experiment exp_name type: gui/batch {…}

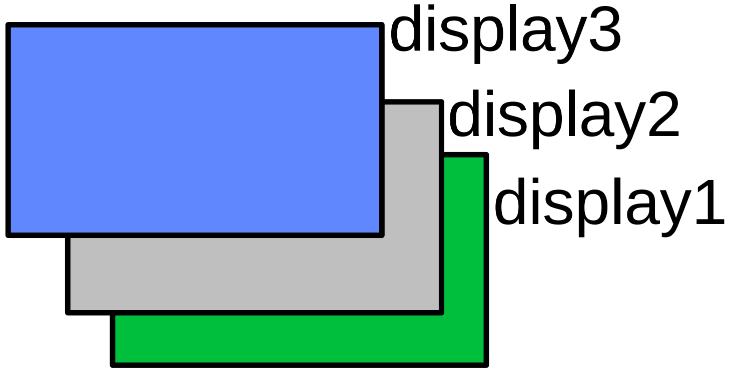
- **gui** : experiment with a graphical interface, which displays its input parameters and outputs.
- **batch** : Allows to setup a serie of simulations (without graphical interface).

More details on the *batch* mode will be given later in the tutorial

Step 1: Outputs of an experiment

- *output* blocks are defined in an *experiment* and define how to visualize a simulation (with one or more *display* blocks that define separate windows)
  - Each *display* can be refreshed independently by defining the facet **refresh_every:** *nb (int)* (the display will be refreshed every *nb* steps of the simulation)
  - Each *display* can include different layers (like in a GIS) :
    - Agents species : **species** *my_species* **aspect:** *my_aspect*
    - Images: **image** *layer_name* **file:** *image_file*;
    - Texts : **texte** *layer_name* **value:** *my_text*;
    - Charts : see later.
    - ...

Warning: in a *display*, the drawing of layers follows the order used in its definition.

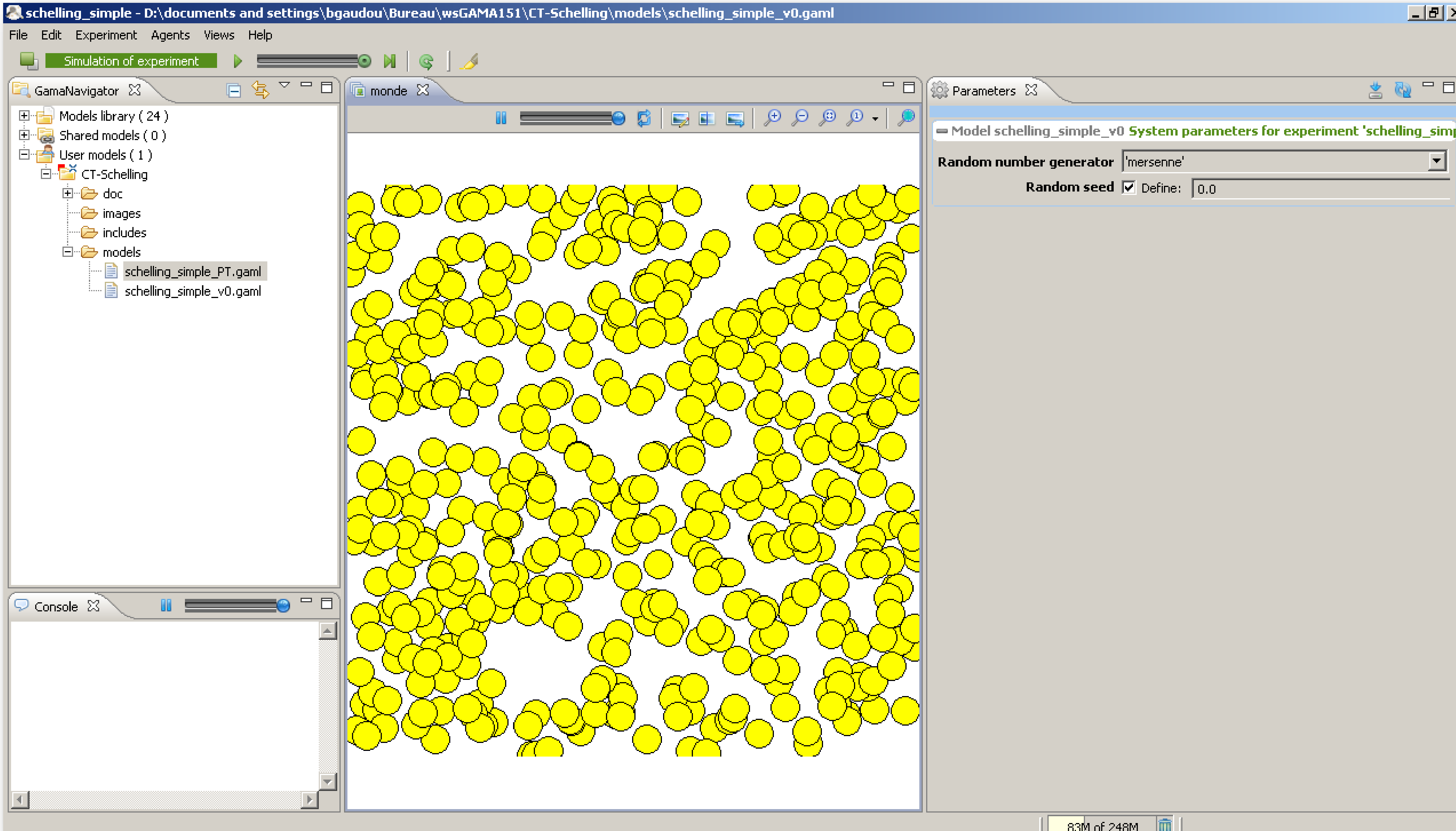


For our model: Definition of one display

```
experiment schelling type: gui {
  output {
    display display_people {
      species people aspect: base;
    }
  }
}
```

```
model CT_Schelling

global {
}

environment {
}

entities {
}

experiment schelling type: gui {
}
```

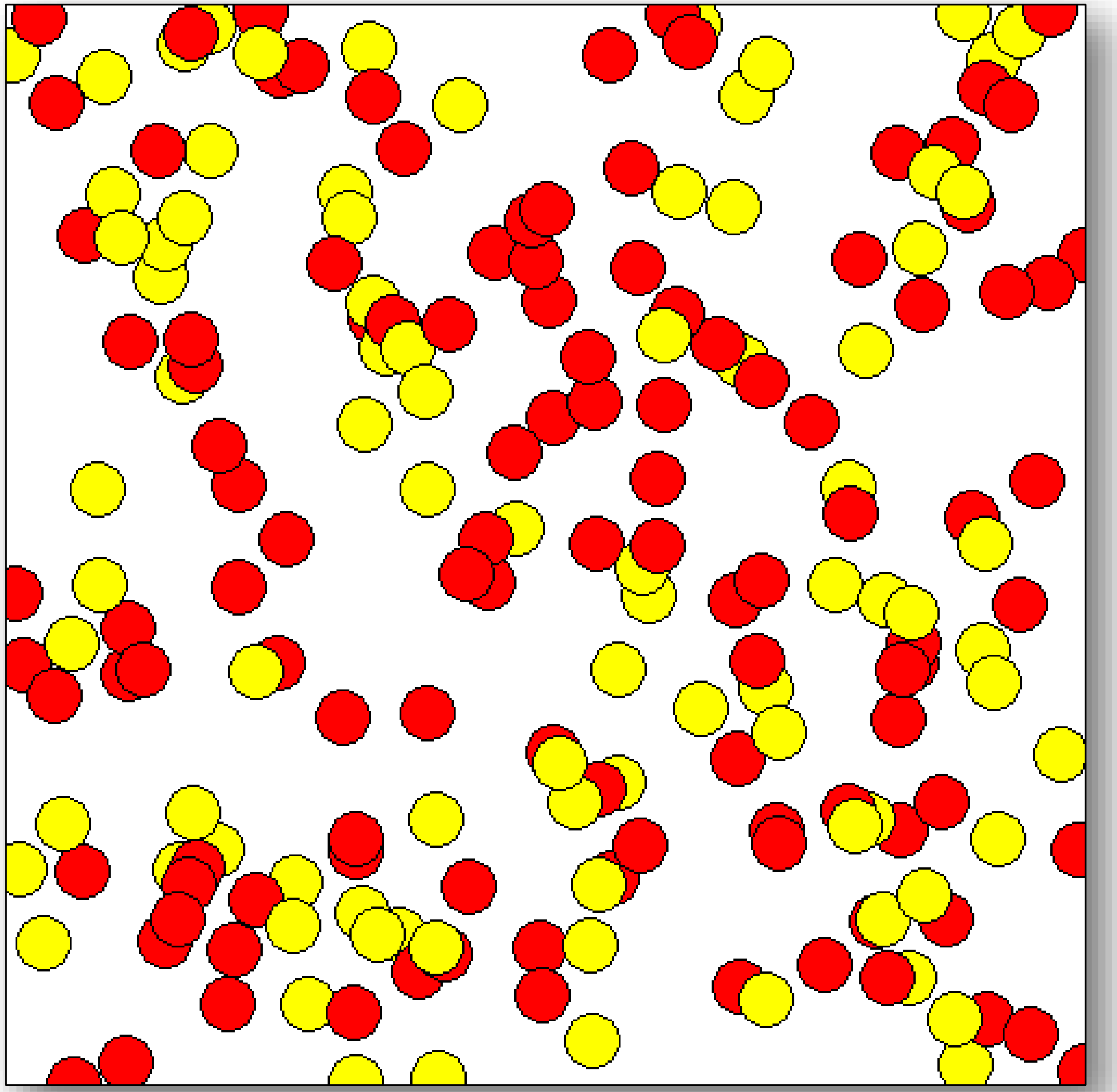
Step 1: End


schelling_simple - D:\documents and settings\bgaudou\Bureau\wsGAMA151\CT-Schelling\models\schelling_simple_v0.gaml
File Edit Experiment Agents Views Help
Simulation of experiment
GamaNavigator
Models library (24)
Shared models (0)
User models (1)
CT-Schelling
doc
images
includes
models
schelling_simple_PT.gaml
schelling_simple_v0.gaml
monde
Parameters
Model schelling_simple_v0 System parameters for experiment 'schelling_simp
Random number generator
'mersenne'
Random seed
Define:
0.0
Console
83M of 248M

Step 2: Creation of the two groups

❖ Objectives:

- Addition of a parameter to change the initial number of agents
- Addition of an attribute to the people species to deal with various colors (and thus groups)



❖ Key points:

- Introduction of parameters
- Introduction of operators
- Introduction of species attributes

Step 2: Introduction of parameters

- To introduce a parameter in a GAMA model, we should follow 2 steps:
  - Introduce a global variable (in the global block)
  - Introduce a parameter statement (in the experiment block)

Variable definition : ***type of the variable*** or ***const*** + *name*

- For constants, a mandatory facet:
    - ***type***: int (integer), float (floating point number), string, bool (boolean, *true* or *false*), point (coordinates), list, pair, map, file, matrix, espèce d'agents, rgb (color), graph, path…
- Optional facets:
    - **<-** (initial value), ***update*** (value recomputed at each step of the simulation), ***function:{..}*** (value computed each time the variable is used), ***min***, ***max***

For our model: Definition of the *nb_people* global variable

```
global {
  int nb_people <- 500 min: 10 max: 1000;
  ...
}
```

Global variable with an initial value

```
model CT_Schelling

global {
}

environment {
}

entities {
}

experiment schelling type: gui {
}
```

Step 2: Introduction of parameters – parameter statement

- Parameter definition

parameter + text + var: + name_variable [+ optional facets]

- text : will be displayed in the GUI to represent the parameter
- name_variable : the variable that will be modified by the user
- Optional facets include : category (parameters are organized into categories in the graphical interface)

For our model: Definition of the *nb_people* parameter

```
experiment schelling type: gui {
  parameter "Number of inhabitants"
var: nb_people category: "people";

  ...
}
```

Definition of the new parameter

```
model CT_Schelling

global {
}

environment {
}

entities {
}

experiment schelling type: gui {
}
```

- It is possible to directly use global variables in the model

For our model: Use of the *nb_people* parameter to define the number of *people* agents to create

```
global {
  int nb_people <- 500 min: 10 max: 1000;
  init {
   create people number: nb_people;
  }
}
```

Use of the nb_people global variable

```
model CT_Schelling

global {
}

environment {
}

entities {
}

experiment schelling type: gui {
}
```

Step 2: customize aspect

- We add a new attribute to people species: agent_color
  - it is a color (type: rgb)
  - initialized with the color red (if flip(0.5) = true), with the yellow color otherwise
  - We use it in the aspect (color of the circle)

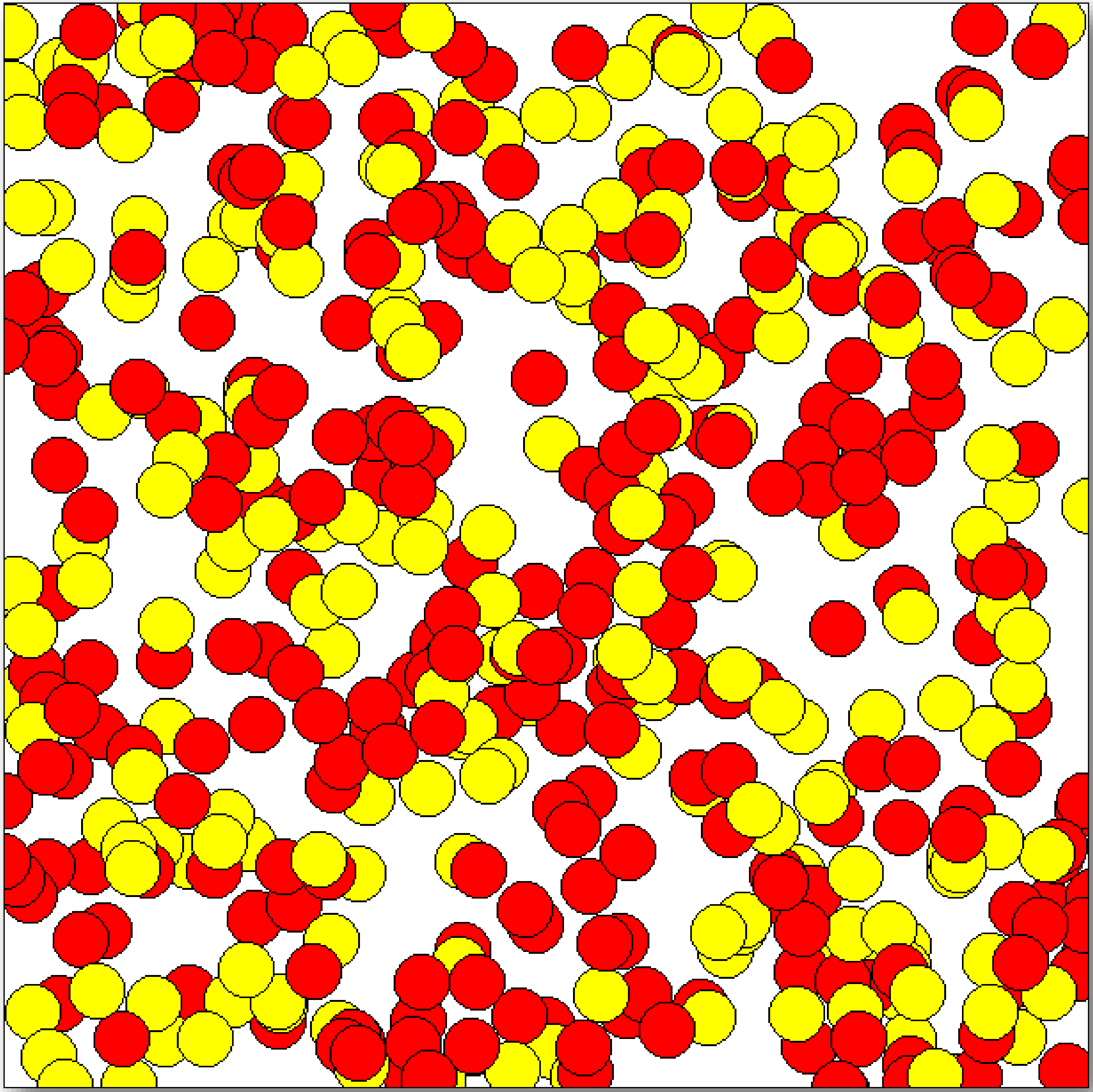
Flip(proba) :
returns true with the probability proba

(cond **?** Value1 **:** value2) : returns value1 if cond is true returns value2 otherwise

```
entities {
  species people {
   rgb agent_color <- flip(0.5) ? rgb("red") : rgb("yellow");

    aspect base {
     draw circle size: 5 color: agent_color;
    }
  }
}
```

Use of the agent_color attribute

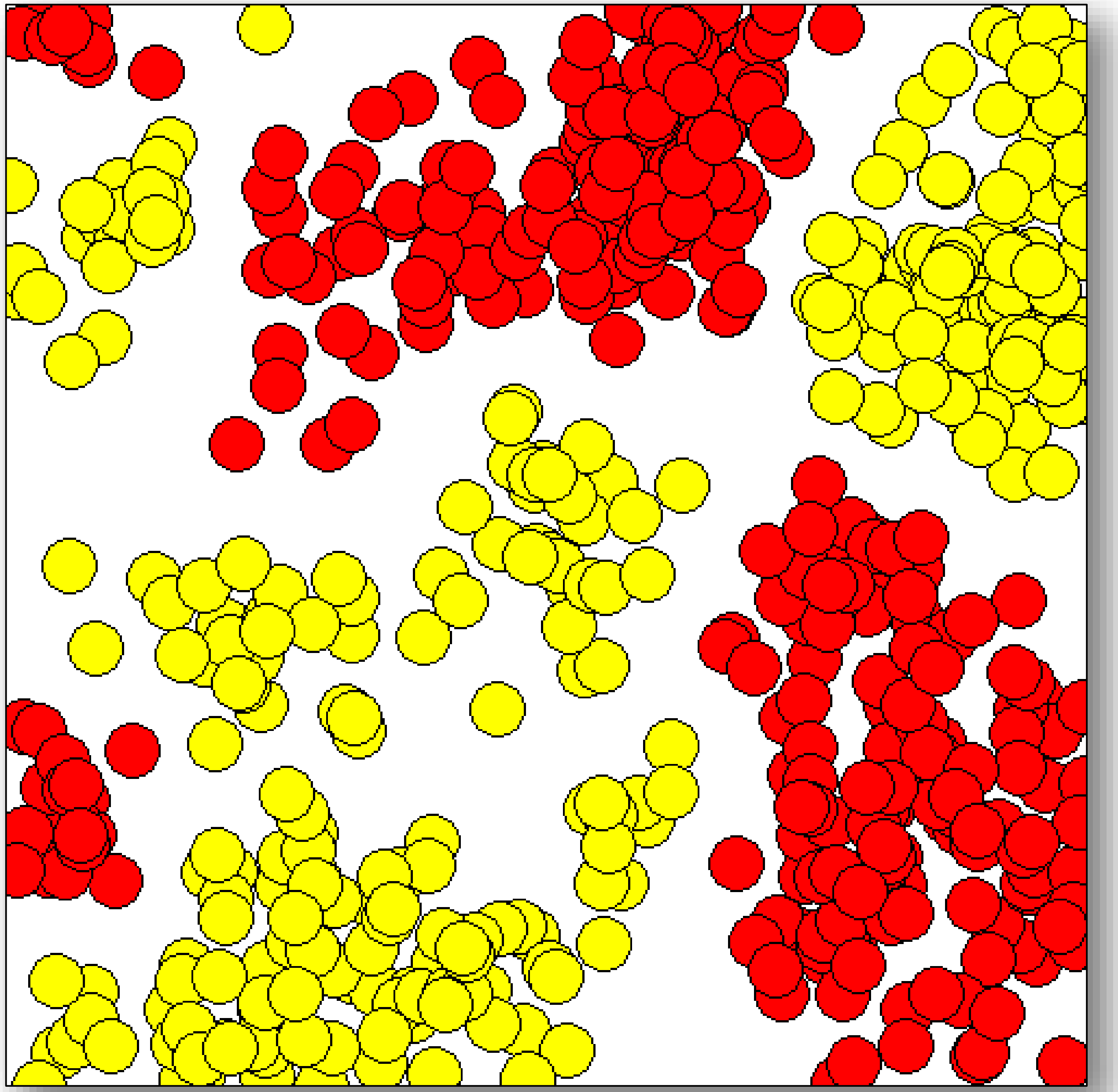
```
model CT_schelling

global {
}

environment {
}

entities {
}

experiment schelling type: gui {
}
```



Step 3: Dynamics of the model

Objective:

- Computation of the state of the agent: happy of not ?
- If the agent is not happy, it will move



Key points:

- Definition of the agent behaviors

Step 3: new attributes for people species

- We introduce two new attributes to people species.
  - preference : the rate of similar agents below which it will not be happy anymore.
  - not_happy : is the people happy or not?

```
entities {
  species people {
    rgb agent_color <- flip(0.5) ? rgb("red") : rgb("yellow");
    float preference <- 0.5;
    bool not_happy <- true;
    ...
}
```

type: <u>float</u>

type: <u>bool</u> (for boolean) = only true or false

GAMA includes a powerful casting conversion system : *type_name(val)* (or *val as type_name*) allows to convert *val* in the new type *type_name*. For instance : *rgb("red")* returns the red color (casting of the *string* 'red' into a value of type *rgb*)

```
model CT_schelling

global {
}

environment {
}

entities {
}

experiment schelling type: gui {
}
```

Use of agent variables (attributes)

- It is possible to access a variable by: *my_agent.my_variable*
- The ***set*** instruction allows to modify the value of a variable
    - ***set** ma_variable* **<-** *nouvelle_valeur**;***

For example: Definition of a *move* action

Operator: Returns a random point of the geometry

Built-in attribute of every agent (geometry of the agent)

```
set location <- any_location_in (world.shape);
```

Built-in attribute of every agent: coordinate of the agents {x,y}

world is the embedding world agent

Allows to modify the value of the agent current location by a location randomly drawn in the geometry of the world agent

Step 3: Defining a new action

- An action is a capability available to the agents of a species (what they can do)
- It is a block of statements that can be used and reused whenever needed
- An action can accept arguments (statement **arg** *nom_arg* **type:** *type*)
- An action can return a result (statement **return**)

Definition of arguments (optional)

Returns a result (optional)

```
action my_action {
    arg arg1 type: int;
    arg arg2 type: int;
    //...
    return arg1 + arg2;
}
```

For our model: Definition of a *move* action

```
entities {
        species people {
                ...
                action move {
                        set location <- any_location_in (world.shape);
                }
        }
}
```

```
model CT_schelling

global {
}

environment {
}

entities {
}

experiment schelling type: gui {
}
```

Step 3: Calling an action

- There are two ways to call an action: using a statement or as part of an expression
- **do** *action_name* **arg1:** *v1* **arg2:** *v2*;
- **set** *my_var* **<-** self *action_name* [**arg1**::v1, **arg2**::v2];

When a value is returned

Denotes the agent that will execute the action (the action must be defined in its species)

Use of a map to pass arguments

```
do write message: "name: " + name;
```

Action name

Argument name

Second argument name

```
set my_variable <- self my_action [arg1::4, arg2::10];
```

Action name

First argument name

Step 3: Definition of a reflex

- A *reflex* is a block of statements (that can be defined in *global* or any species) that will be automatically executed at each time step of the simulation if its condition is *true.*
- **reflex** *reflex_name* **when:** *condition* {…}
- The **when** facet is optional: when it is ommitted, the *reflex* is activated at each time step.

Reserved keyword indicating the value of the current time step

```
reflex my_reflex when: time = 5 {

}
```

This reflex will only be activated during the fifth step of the simulation be

Step 3: species - behavior definitions

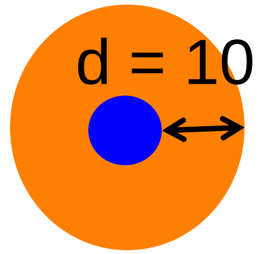
1st behavior: the agent compute its happiness:

- He will get the list of his neighbors:
- He will count the number of neighbors with the same color
- He will count the total number of neighbors
- He will compare with his preference rate to determine whether he is happy

New statements

- **let** new_var **type:** type **<-** value**;**
  Create a new local variable of given type and affect value; A local variable is a variable that has an existence only in a statement block: as soon as the end of the block, the variable is deleted from the computer memory

- **1st behavior:** the agent computes its happiness:




```
entities {
  species people {
    ...
    reflex update_state {
      let neighbours type: list of: people <- people at_distance 10;
      let nb_same_color type: float <-
          neighbours count (each.agent_color = agent_color);
      let nb_total type: float <- length(neighbours);
      set not_happy <- nb_same_color / nb_total < preference;
    }
    ...
  }
}
```

Returns the list of people agents at a distance of 10 from the agent calling it

list count (cond) Count the number of item of the list satisfying the condition

length(list) returns the number of element of the list

```
model CT_schelling

global {
}

environment {
}

entities {
}

experiment schelling type: gui {
}
```

Manipulating lists

- GAMA offers numerous operators to manipulate lists and containers

- Unary operators : *min*, *max, sum…*

- Binary operators :
  - *where* : returns a sub-list where all the elements verify the condition defined in the right operand.
  - *first_with* : returns the first element of the list that verifies the condition defined in the right operand.
  - …

- In the case of binary operators, each element can be accessed with the keyword ***each***

```
[2, 5, 3, 2, 20] where (each > 3)
```

⇨ [5, 20]

```
[2, 5, 3, 2, 20] first_with (each > 3)
```

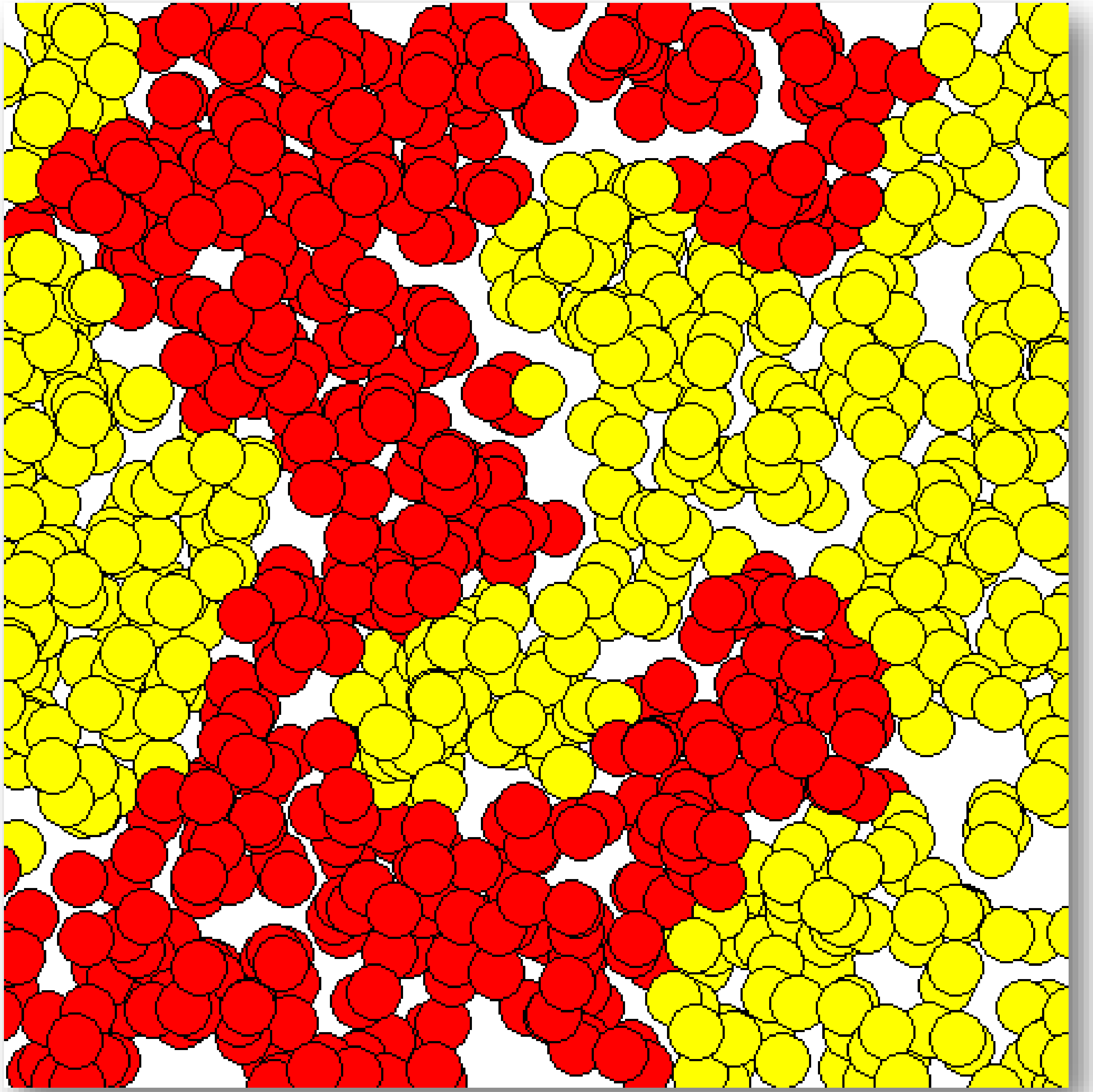
⇨ 5

2nd behavior: if he is not happy, he will move

```
entities {
  species people {
    ...
    reflex movement when: not_happy {
     do move;
    }
  }
}
```

calling of the move action

```
model CT_schelling

global {
}

environment {
}

entities {
}

experiment schelling type: gui {
}
```



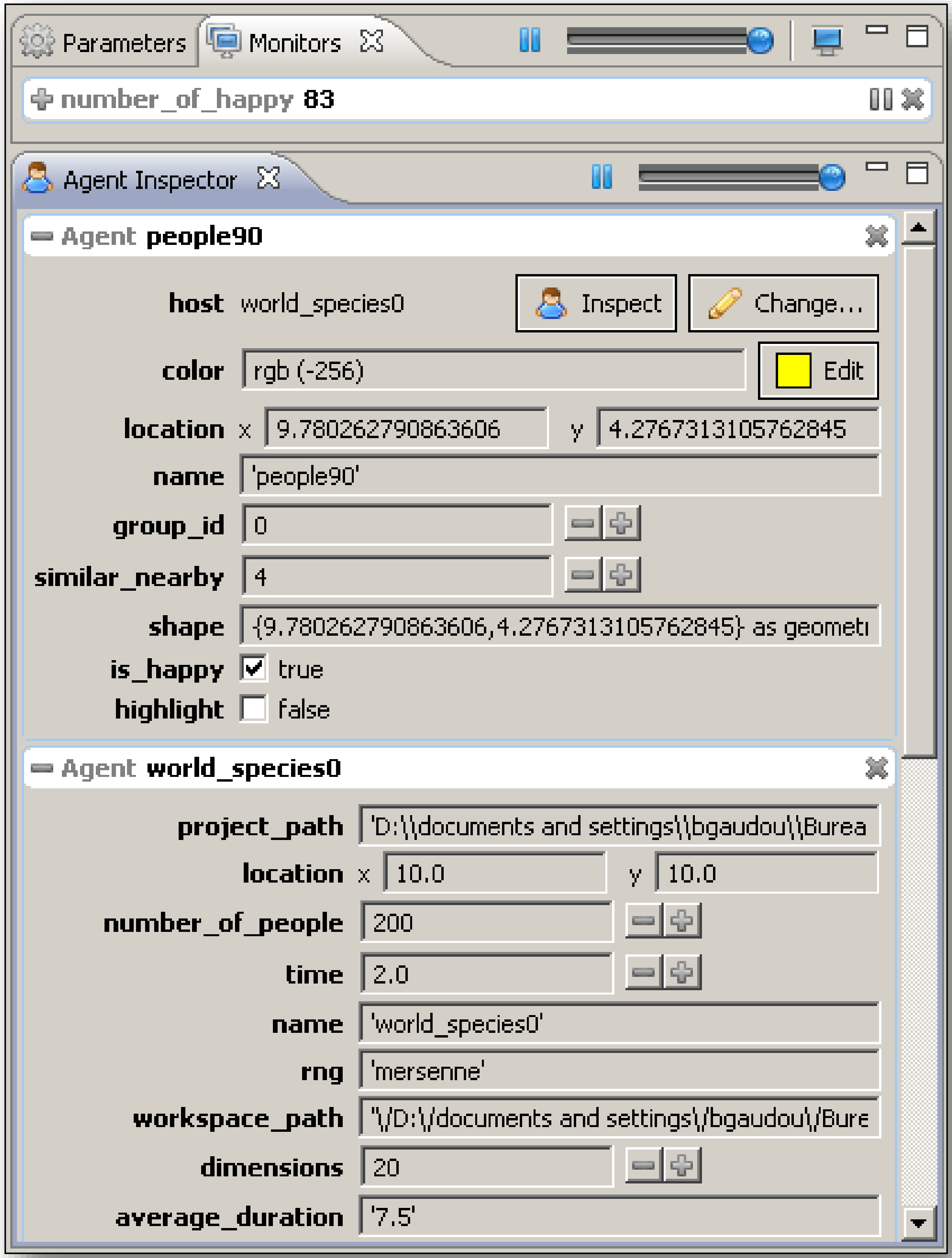
Step 4: monitor and inspect agents

❖ Objective:

- Adding of a monitor to follow the evolution of the number of *happy* agents

❖ Key points:

- Monitor definition



```
model CT_Schelling

global {
}

environment {
}

entities {
}

experiment schelling type: gui {
}
```

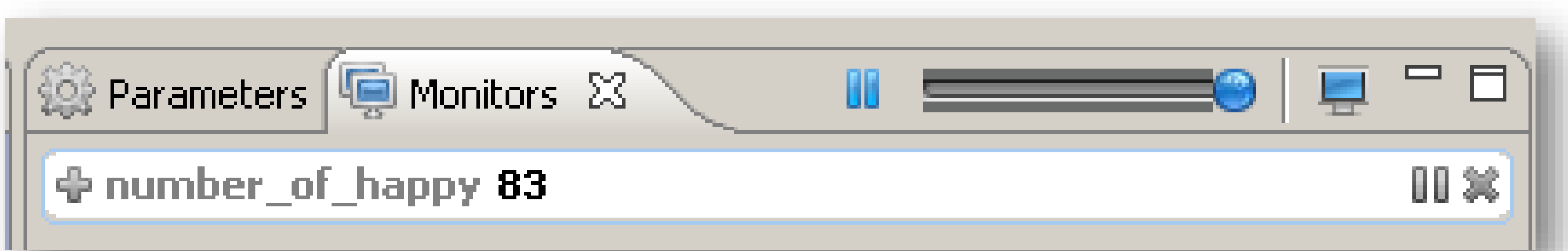
- Allows to follow the value of a GAML expression:

monitor name value: expression;

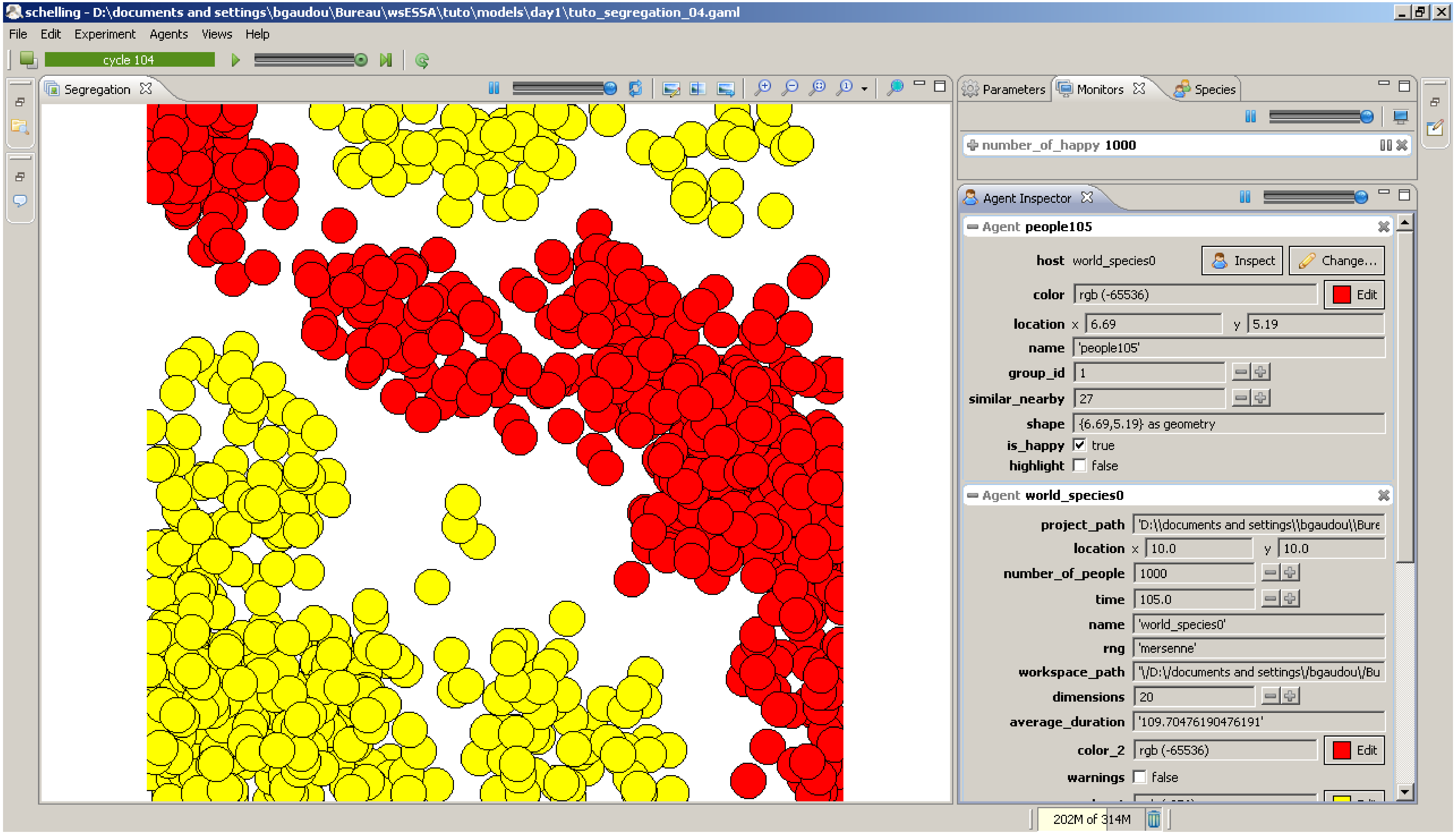
value: mandatory, expression computed

For our model: Definition of a *monitor* to follow the number of happy people agents

```
experiment schelling type: gui {
  ...
  output {
   ...

   monitor 'number_happy'
value: length(list(people) where (each.not_happy = false));
  }
}
```



Model 4: End


schelling - D:\documents and settings\bgaudou\Bureau\wsESSA\tuto\models\day1\tuto_segregation_04.gaml
File Edit Experiment Agents Views Help
cycle 104
Segregation
Parameters
Monitors
Species
number_of_happy 1000
Agent Inspector
Agent people105
host world_species0
Inspect
Change...
color rgb (-65536)
Edit
location x 6.69 y 5.19
name 'people105'
group_id 1
similar_nearby 27
shape {6.69,5.19} as geometry
is_happy true
highlight false
Agent world_species0
project_path 'D:\\documents and settings\\bgaudou\\Bure
location x 10.0 y 10.0
number_of_people 1000
time 105.0
name 'world_species0'
rng 'mersenne'
workspace_path '\/D:\/documents and settings\/bgaudou\/Bu
dimensions 20
average_duration '109.70476190476191'
color_2 rgb (-65536)
Edit
warnings false
202M of 314M

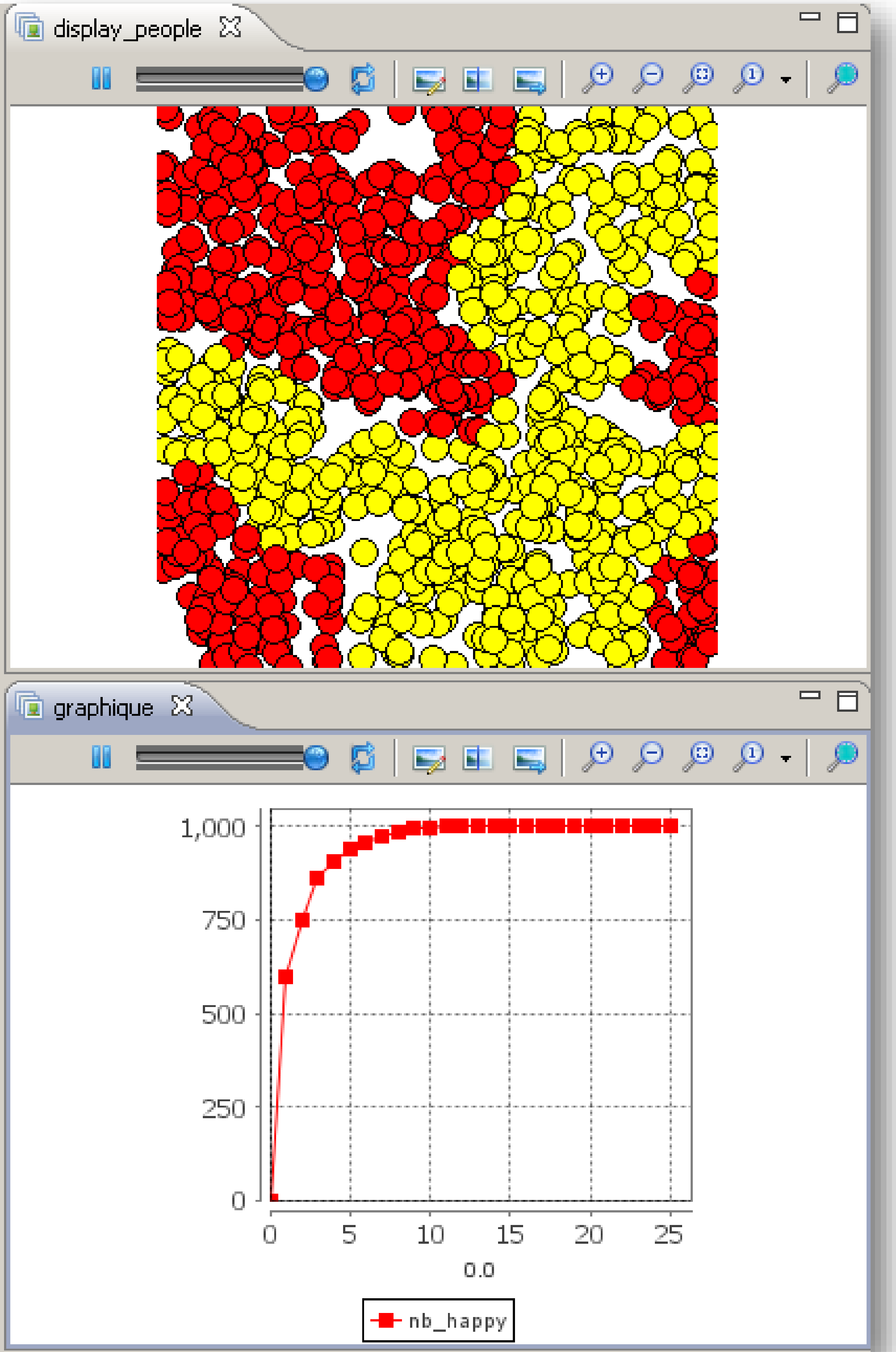
Model 5: Addition of a chart

Objective:

- Addition of a new display to visualize:
    - A serie representing the number of happy people

Key points:

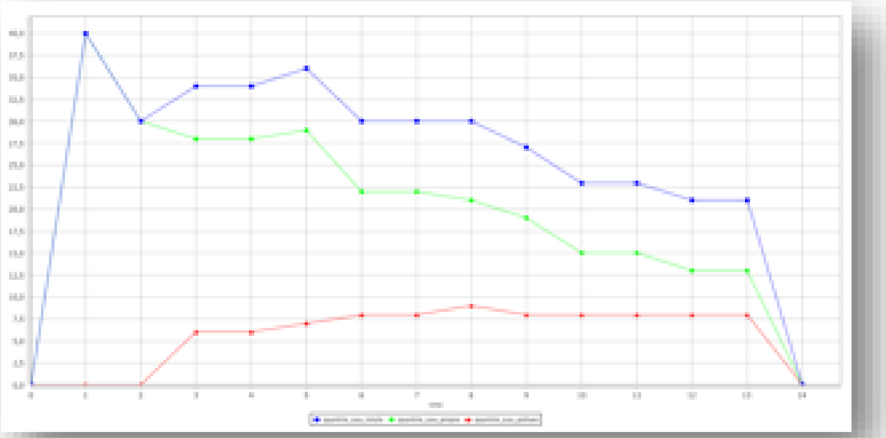
- Definition of charts



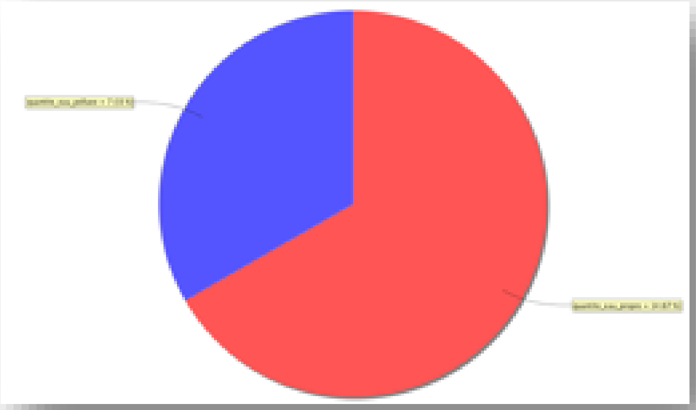
Definition of charts

❖ GAMA can display various chart types:

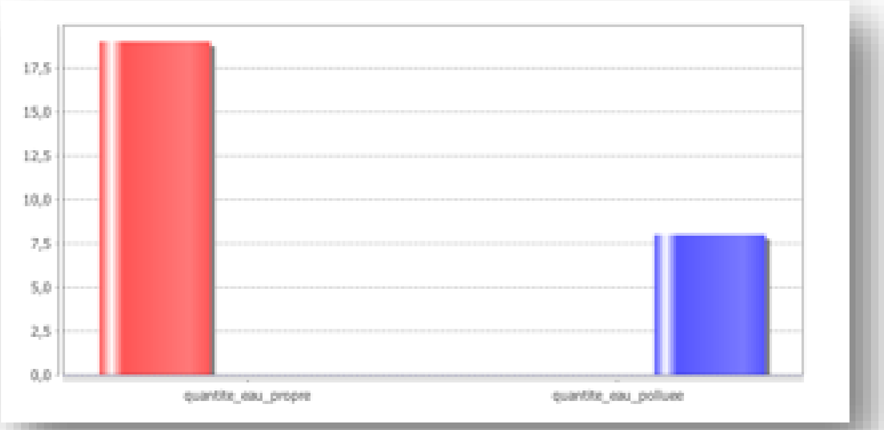
- Time series



- Pie charts



- Histograms



Model 5: Experiment – new chart display

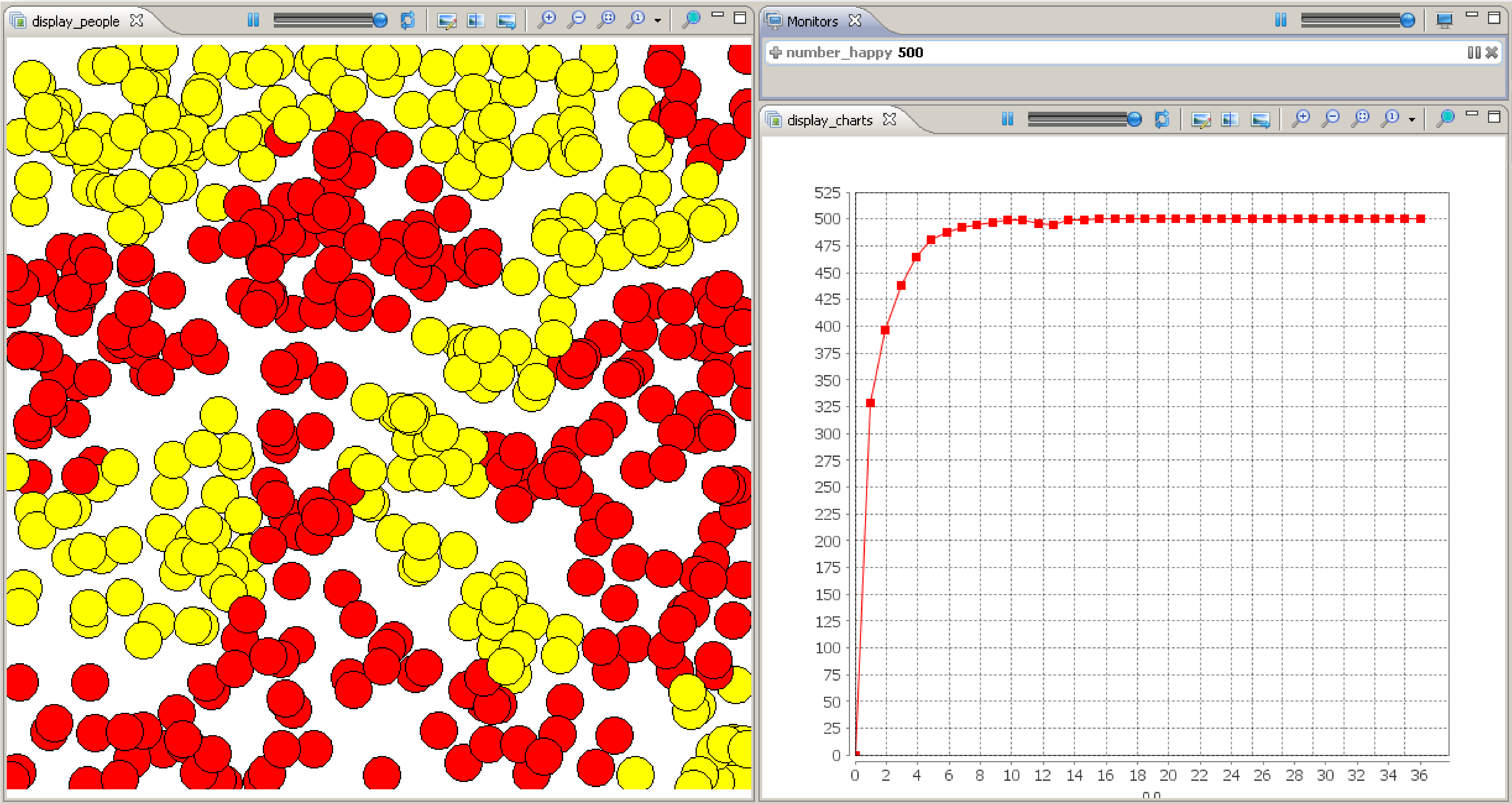
- Definition of a new display with one series chart

For our model: Definition of a *chart* to follow the evolution of several variables

```
experiment schelling type: gui {
  ...
  output {
   ...
    display display_charts {
      chart "nb_happy" type: series background: rgb("white") {
         data "nb_happy" color: rgb("red")
            value: list(people) count (! each.not_happy);
      }
    }
  }
}
```

```
model CT_Schelling

global {
}

environment {
}

entities {
}

experiment schelling type: gui {
}
```

Model 5: end


display_people
Monitors
number_happy 500
display_charts
525
500
475
450
425
400
375
350
325
300
275
250
225
200
175
150
125
100
75
50
25
0
0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36