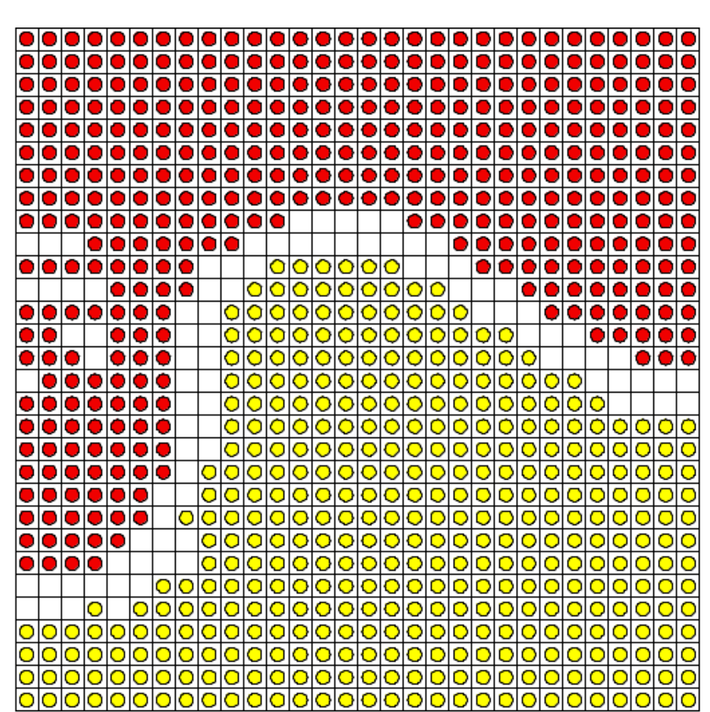
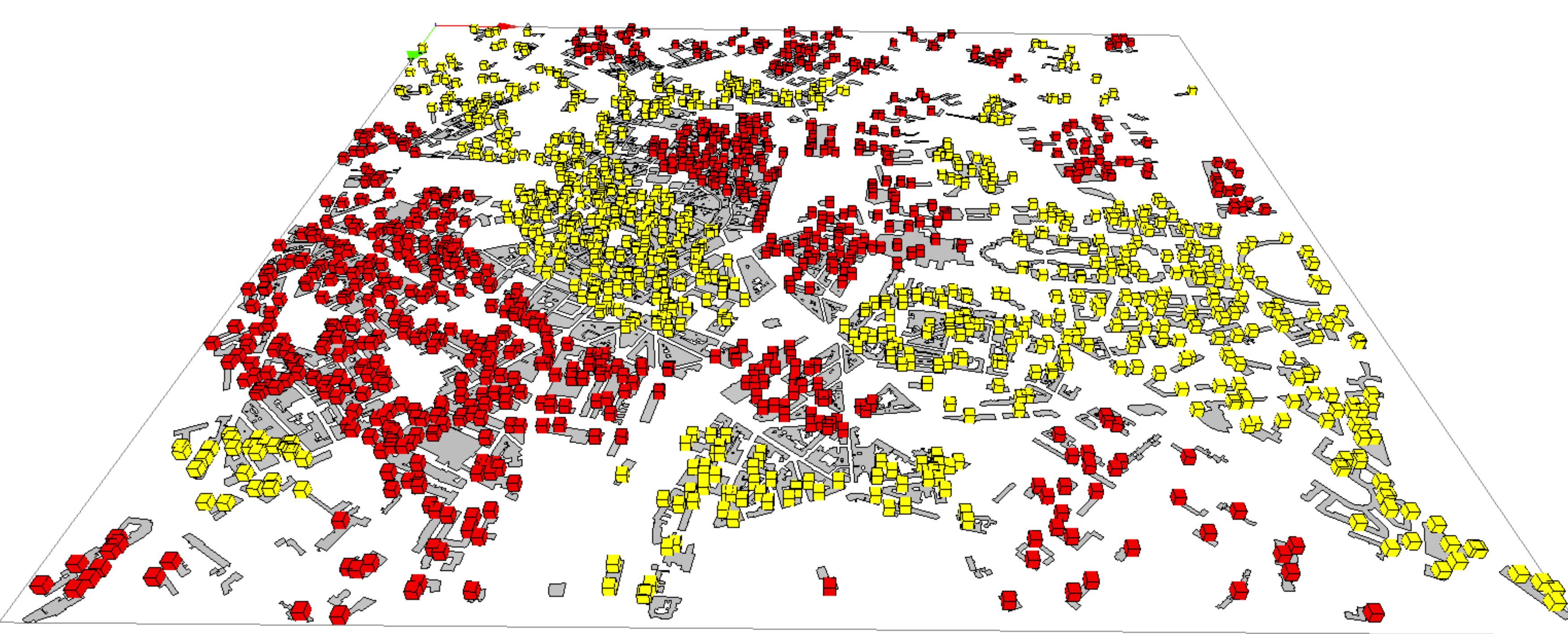
MISS ABMS 2014





GAMA platform: exercice 1 - Schelling

Alexis Drogoul *(a, b)*, **Benoit Gaudou** *(c)*, **Patrick Taillandier** *(d)*

(a) UMI 209 UMMISCO, IRD / UPMC

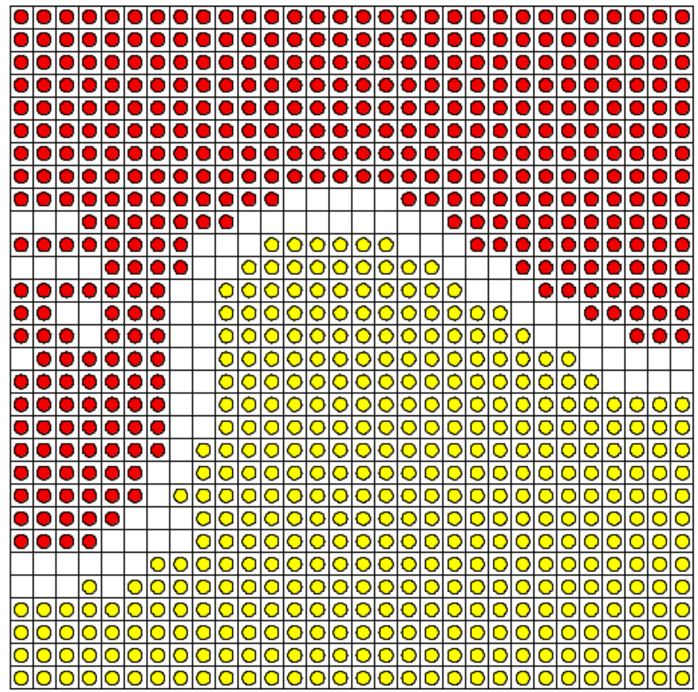
(b) JEAI DREAM, IRD / Université de Can Tho

(c) UMR 5505 IRIT, Université de Toulouse 1 / CNRS

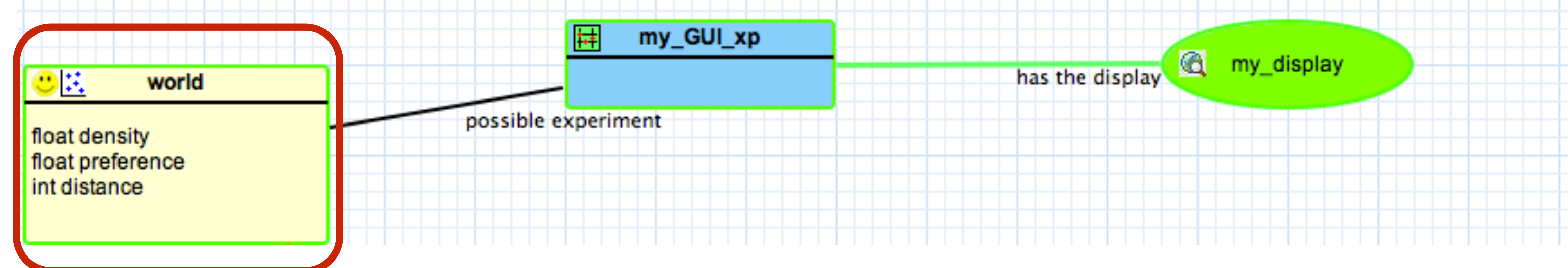
(d) UMR 6266 IDEES, Université de Rouen / CNRS

- 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.

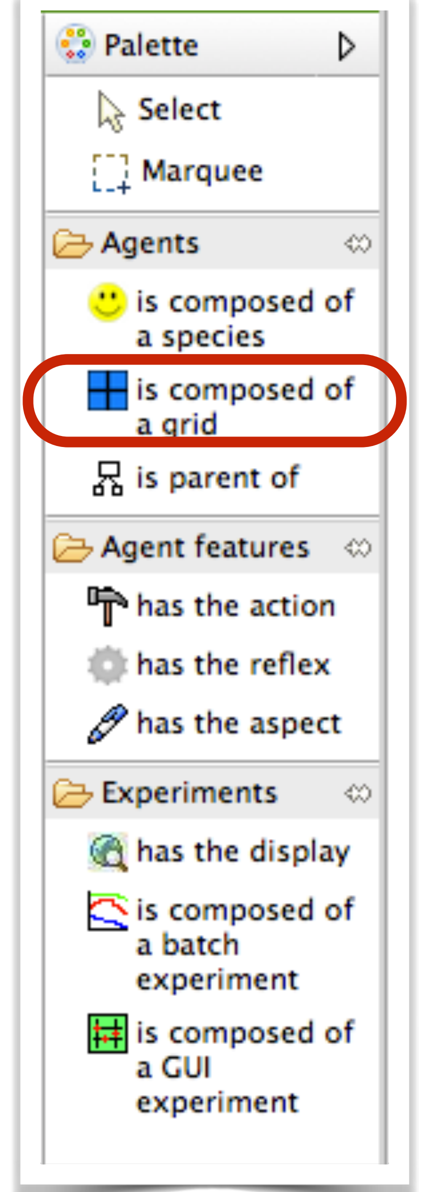
- A grid of cells, inhabited by entities of two different colors.
- Each entity is able to compute the number of neighbors of different color it has around
- At each time step, if the actual percentage computed is higher than its preferred percentage, it moves to another free cell, chosen randomly.



- **Exercice 1**: In the world agent defines 3 new variables :
  - density: type: float, init value: 0.9
  - preference: type: float, init value: 0.7
  - distance: type: int, init value: 5



- **Exercice 2**: define inside the world a new grid called « house » :
  - with 30 rows and 30 columns,
  - with a Moore neighborhood(8)
  - with a variable called « is_empty » with the init value true



- **Exercice 3**: in the display « my_display » add a new layer called « Grid », in which the grid « house » is displayed with black lines

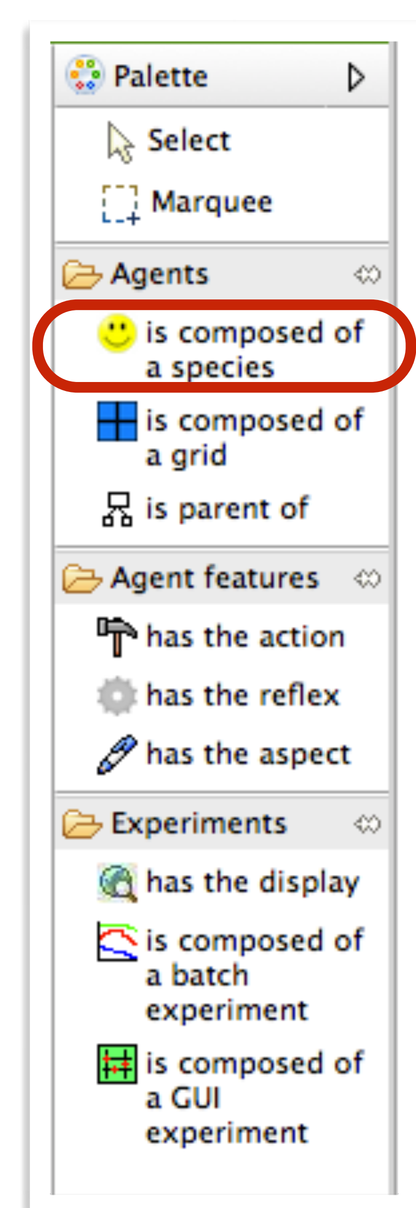
- **Exercice 4**: define inside the world a new species called « people » with 3 variables:
  - my_house: type: house
  - color: type: rgb, init value: flip(0.5) ? #red: #yellow
  - is_happy: type: bool

Probability *0.5* to be red and 0.5 to be yellow

The symbol **#** allows to define a color

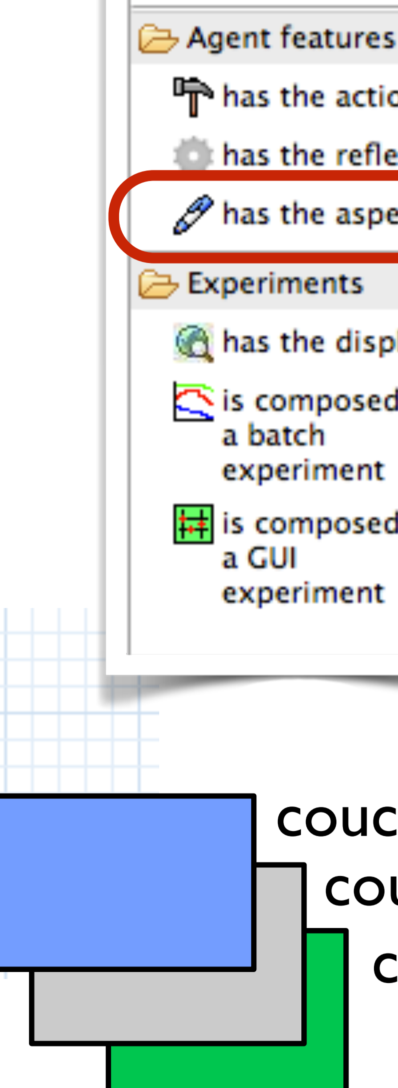
The **flip(proba)** operator is used to test a probability: returns *true* with a probability *proba* (*false* otherwise)

The *condition* **?** *val1* **:** *val2* operator returns *val1* if *condition* is *true*, *val2* otherwise



- **Exercice 5**: Add an aspect to the *people* species called « default »:
  - Add a layer called « People » that draws a *circle* of radius *1.0* with for color the « *color* » expression (i.e. the color variable of the people agents).

- **Exercice 6**: in the display « my_display » add a new layer called « People », in which the species « people » is displayed with aspect « default » (the one previously defined)



world
float density
float preference
int distance
possible experiment
my_GUI_xp
has the display
my_display

Note: in a *display* or a *aspect*, the display order of the layer follows the layer definition

- **Exercice 7**: In the *init* section of the world agent:
- compute the number of people agents to create (nb of house cells x density) :

```
int nb_people <- length(house) * density;
```

- for nb_people *house* cells (chosen randomly), create a people agent that will have for house (*my_house* variable) this house, for location, the house location and set the *is_empty* variable of the house to false:

```
loop h over: nb_people among shuffle(house) {
   create people {
      my_house <- h;
      location <- h.location;
      h.is_empty<- false;
   }
```

The **<-** statement allow to modify the value of a variable

The **length(***list***)** operator returns the nb of elements of a list

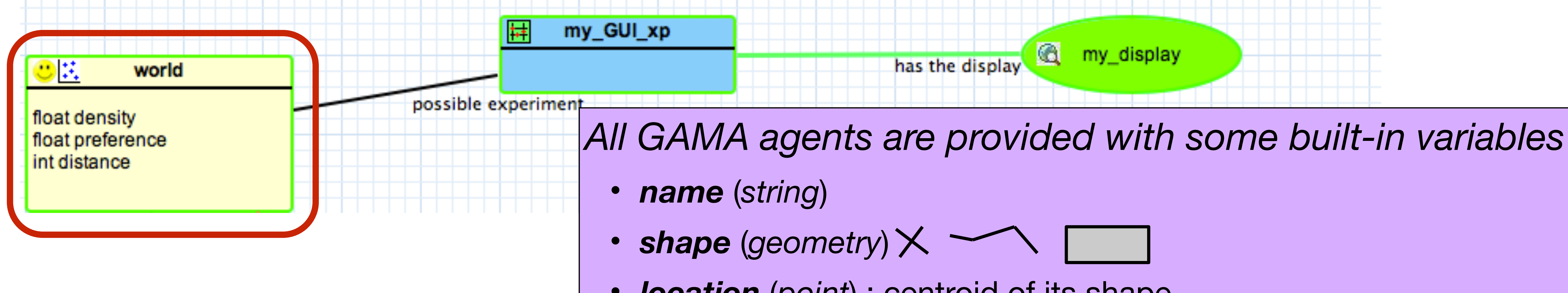
The **loop** *var* **over:** *list* **{...}** statement allows to applied a sequence of statement over the element of a *list*: *var* represents each element of the list

The **create** *a_species* **{...}** statement allows to create a agent of species *a_species*

The **shuffle(***list***)** operator allow to shuffle a list

The *nb* **among** *list* operator allow to randomly draw *nb* element from a list

The *a_agent***.***variable* symbol can be used to access a variable of an agent



All GAMA agents are provided with some built-in variables

- ***name*** *(string)*
- ***shape*** *(geometry)*
- ***location*** *(point)* : centroid of its shape

Step 6: people compute happiness reflex

- **Exercice 8**: Add a reflex to the *people* species called « compute_happiness »:
  - compute the list of *people* agents that are located within the distance *distance* to the agent (its neighborhood):

```
list<people> neigbours_pp <- people at_distance distance;
```

  - compute the number of people agent in the neighborhood:

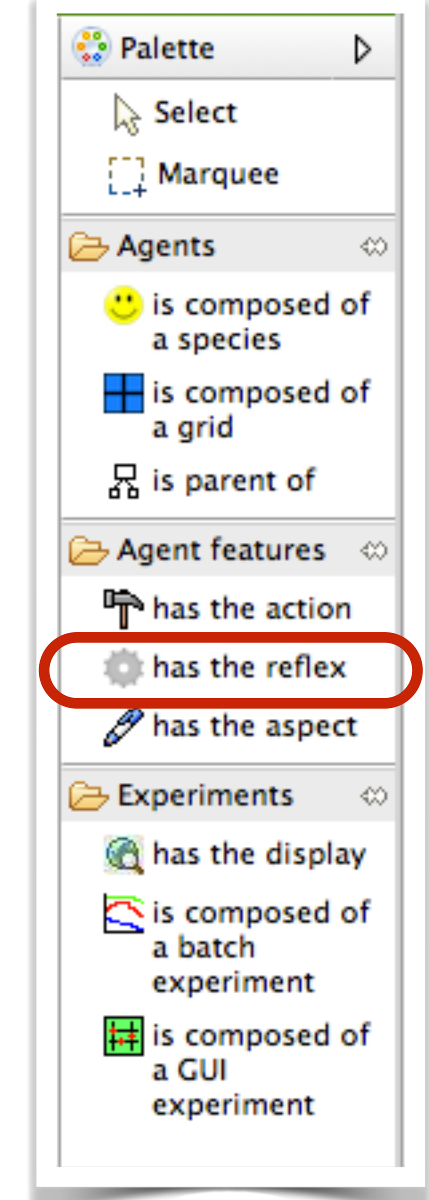
```
int nb_neighbours <- length(neigbours_pp);
```

  - compute the number of *people* agents in the neighborhood that have a different color:

```
int nb_neighbours_diff<- neigbours_pp count (each.color != color);
```

  - compute the *is_happy* variable: the people agent is happy there is no one in its neighborhood or if the rate of people with a different color is lower than its *preference*:

```
is_happy <- (nb_neighbours = 0) or ((nb_neighbours_diff /
nb_neighbours) < preference) ;
```



The *a_species* **at_distance** *distance* operator allows to returns the list of agents of species *a_species* at a distance equal or inferior to *distance* to the agent	The *list* **count** *condition* operator allows to compute the number of elements of the list that verifies the condition (the « *each* » keyword represents each element of the list)

Step 7: people move reflex

- **Exercice 9**: Add a reflex to the *people* species called « move »:
  - add a condition (*Condition*) to the reflex activation: the reflex is activated only if the agent is not happy:

```
not is_happy
```

  - Concerning the reflex gamp code, first, set the variable *is_empty* of *my_house* to *true* (because the agent is leaving it):

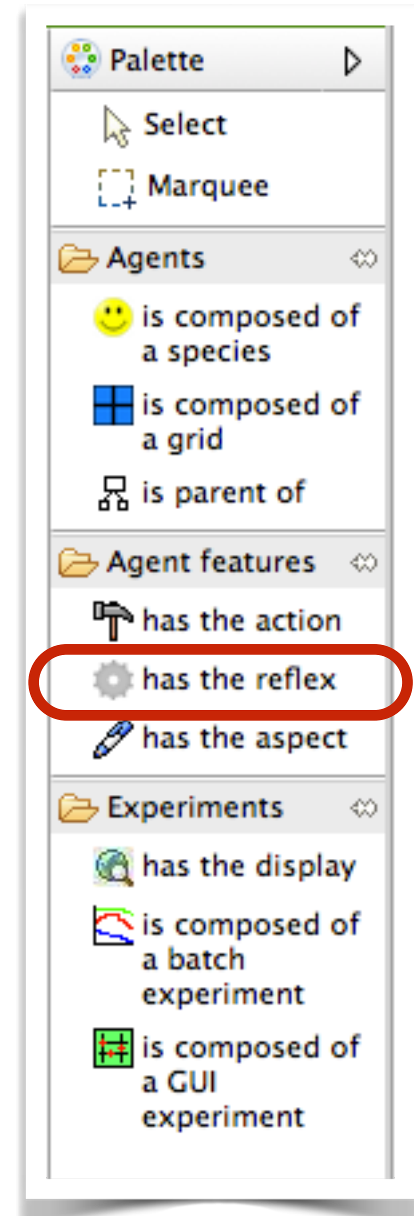
```
my_house.is_empty <- true;
```

  - then, set the variable *my_house* to one empty house:

```
my_house <- one_of (house where each.is_empty);
```

  - then, set the variable *location* to the *my_house* location:

```
location <- my_house.location;
```

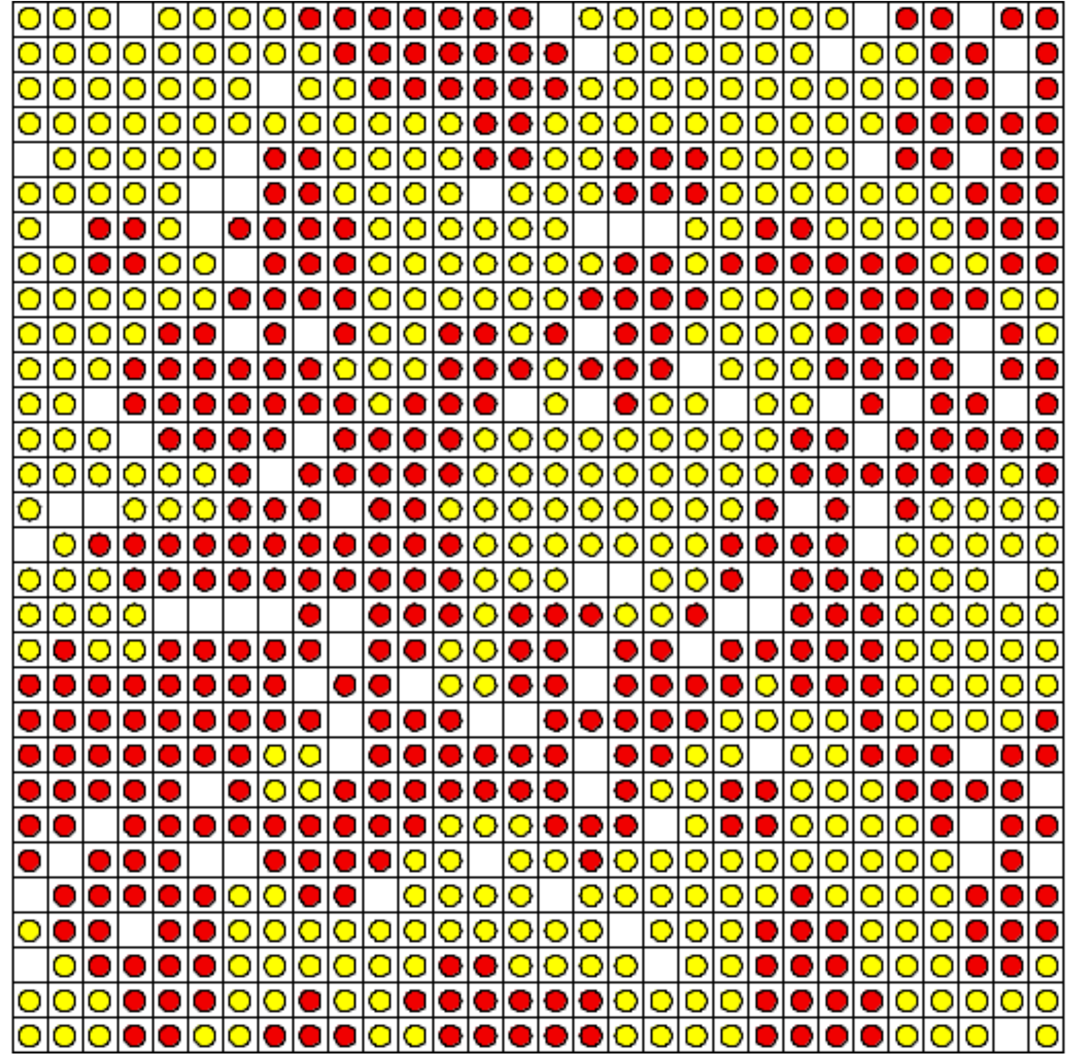


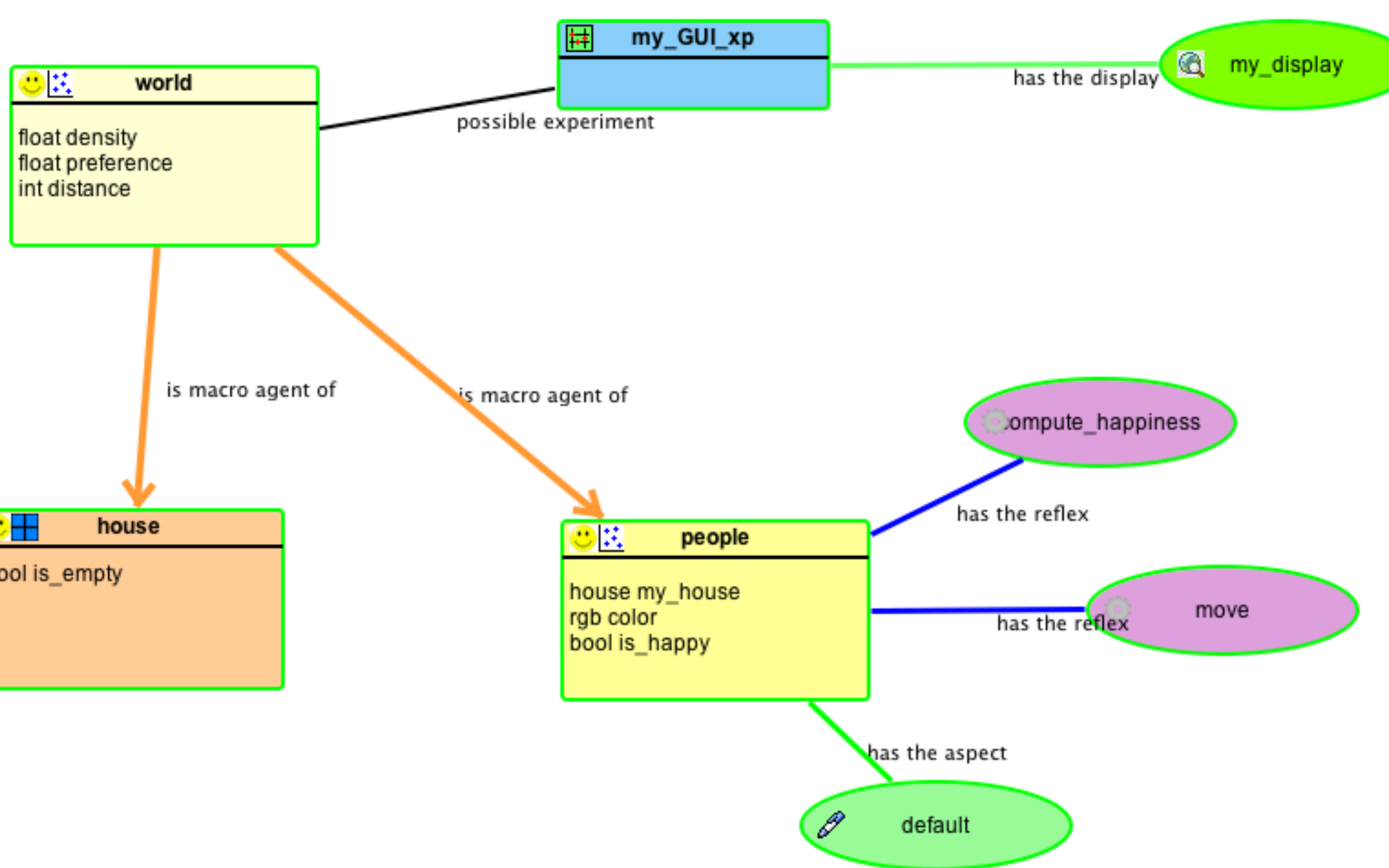
  - finally, set the variable *is_empty* of *my_house* to *false* (because the agent is arriving in the house):

```
my_house.is_empty <- false;
```

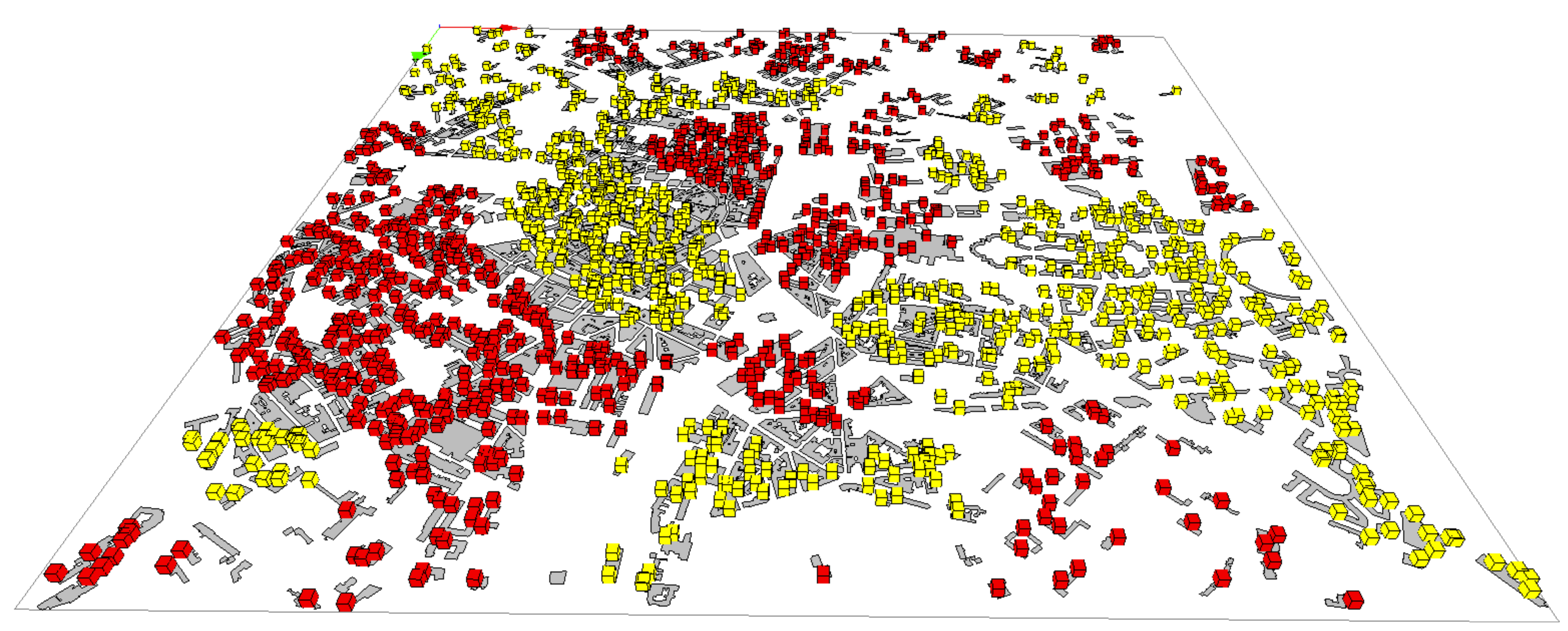
The *list* **where** *condition* operator allows to compute the sub-list of the list that verifies the condition (the « *each* » keyword represents each element of the list)

Conclusion of model 1: it is already finished!



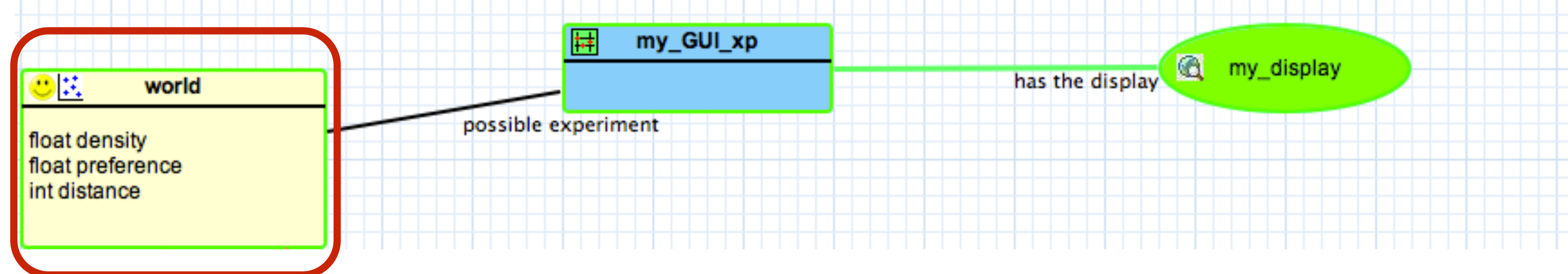


Model2: schelling GIS

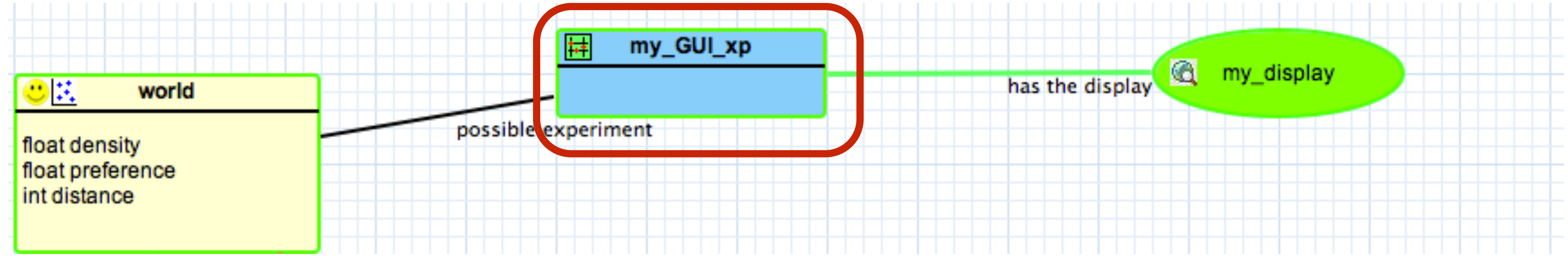


Same model (almost), but with GIS data

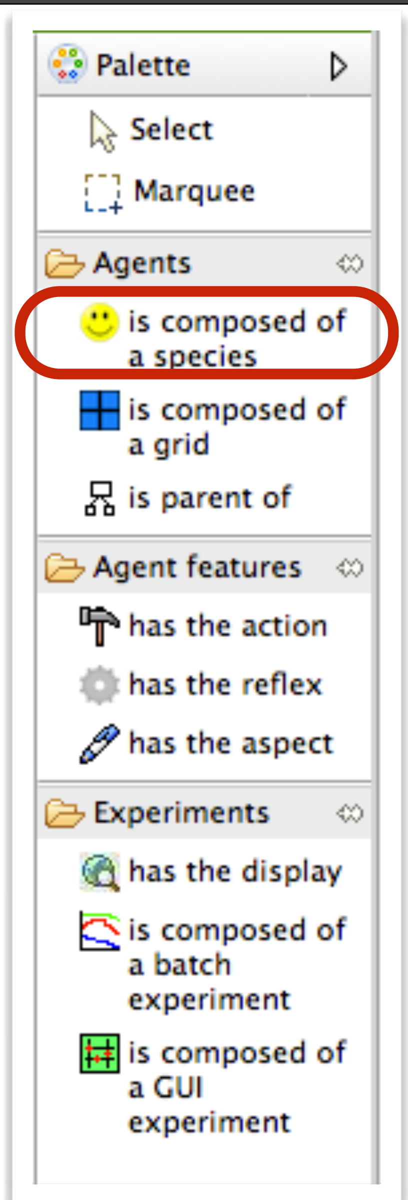
- **Exercice 1**: In the world agent defines 3 new variables :
  - density: type: float, init value: 0.9
  - preference: type: float, init value: 0.5
  - distance: type: int, init value: 100



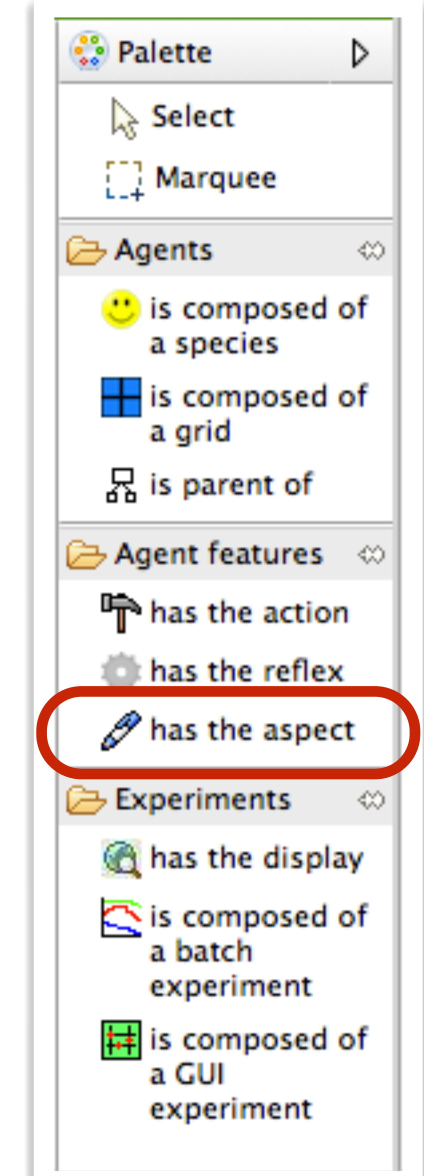
- **Exercice 2**: In the *my_GUI_xp* experiment defines 3 new parameters :
  - density: text: density
  - preference: text: preference
  - distance: text: distance



- **Exercice 3**: define inside the world a new species called « house » :
- ➡ with 1 variable : capacity : type: int (nb of people agents that can come in the house)



- **Exercice 4**: Add an aspect to the *house* species called « default »:
  - Add a layer called « geom » that draws the geometry of the agent with a gray color: choose for Shape type *expression* and then, in the Expression field write *shape* (the *shape* variable is a built-in variable that represents the geometry of the agent).



- **Exercice 5**: in the display « my_display » add a new layer called « House », in which the species « house » is displayed with aspect « default » (the one previously defined). set refresh to *false* (the layer does not need to be redraw every simulation step). Set the display type to *opengl* (to be able to display 3D)

world
float density
float preference
int distance
possible experiment
my_GUI_xp
has the display
my_display

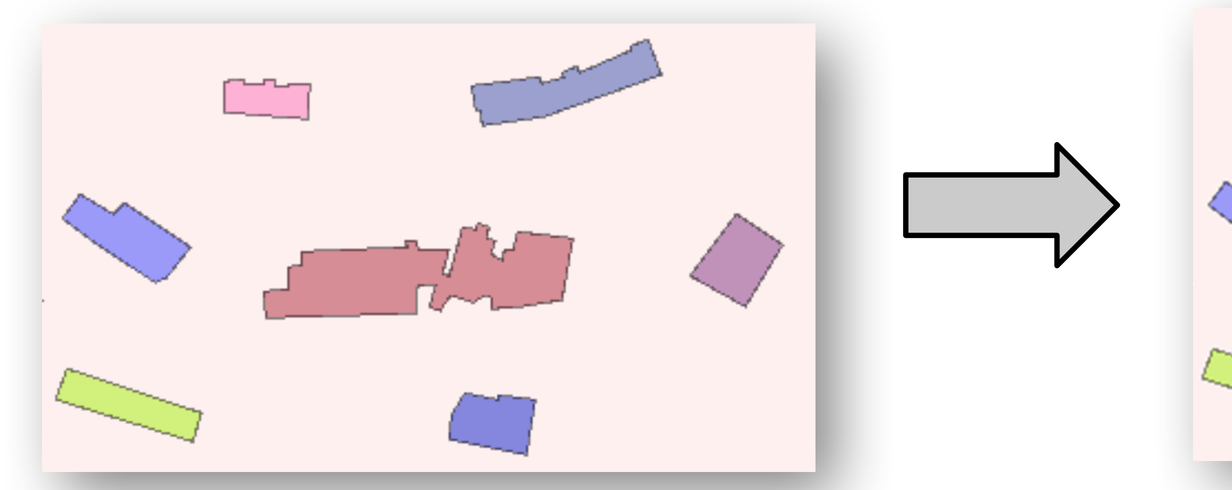
- **Exercice 6**: In the *init* section of the world agent:
  - create house agents from the shapefile buildings.shp that is located in the folder includes
  - For each house agent, set its capacity as 1 + its area/ 1000

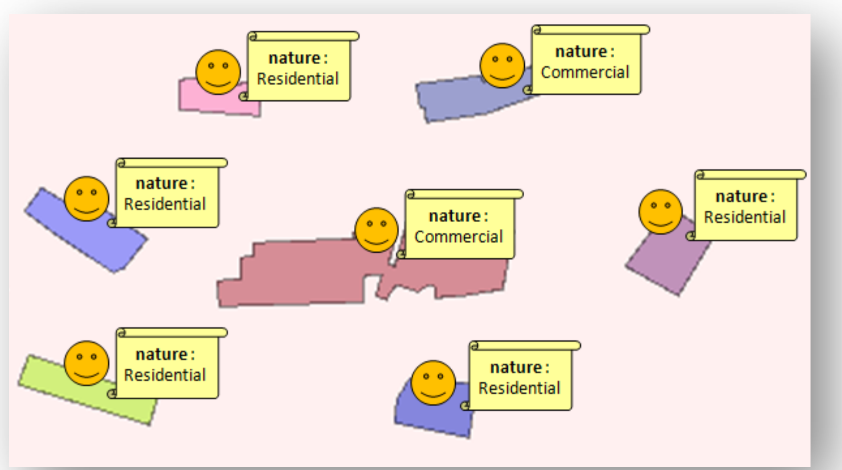
```
create house from: "../includes/buildings.shp" {
    capacity <- 1 + shape.area / 1000;
}
```

It is possible to directly create agents from a shapefile (or from an OSM file) by using the **from** facet with the **create** statement: each object of the GIS file will become an agents

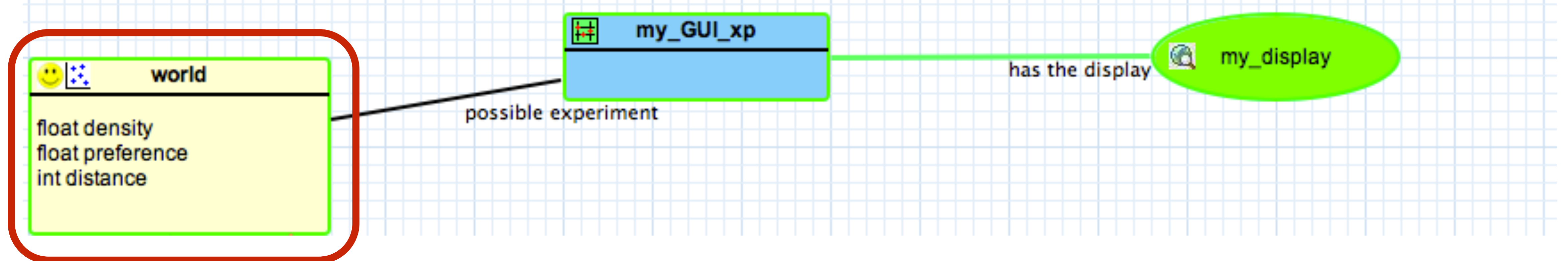
Note that the attribute of the GIS object can be read as well

A geometry has also variables that can be access by *my_geom.variable*: area, perimeter, points…





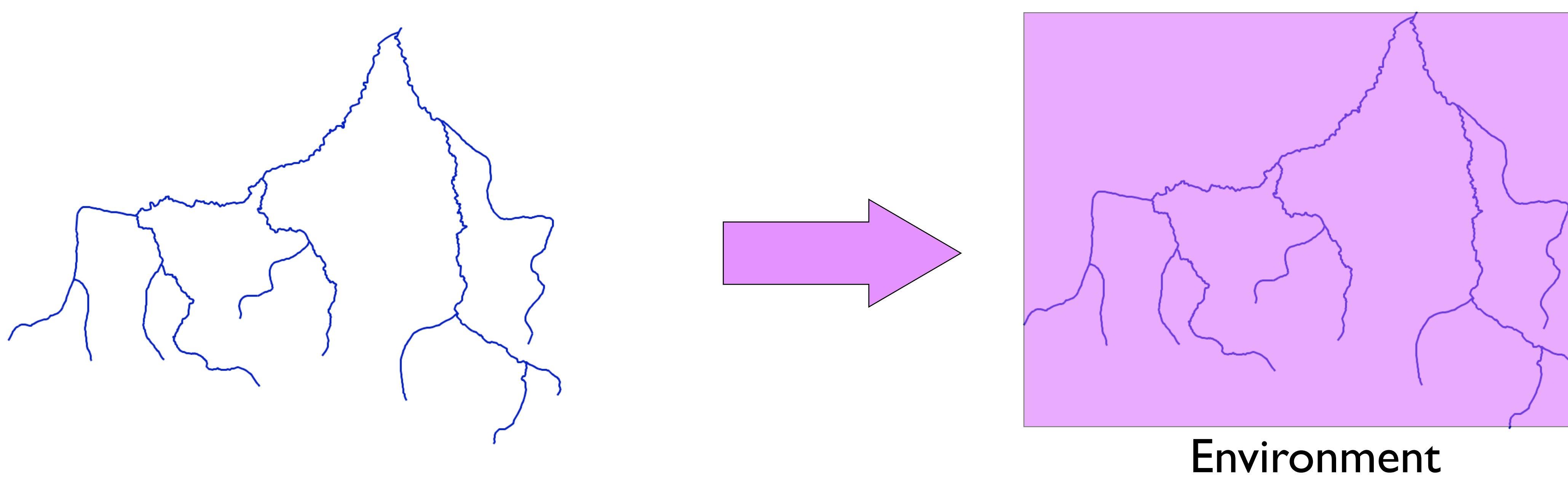
the default folder to consider a file is the diagram/model folder. To go up a level, use « ../ »

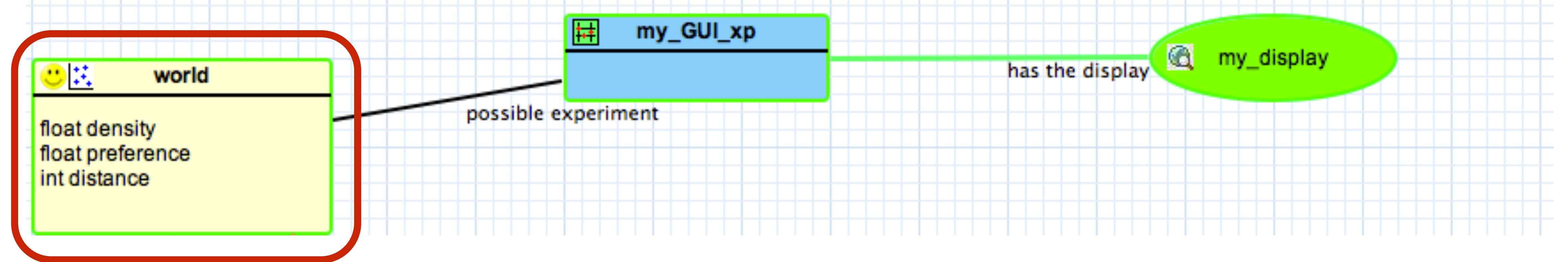


Step 5: definition of the shape of the global

- **Exercice 7**: In the *bounds* section of the world agent:
  - choose the « file » type to define the bounds size of the world
  - As path, choose the buildings.shp shapefile that is located in the folder includes

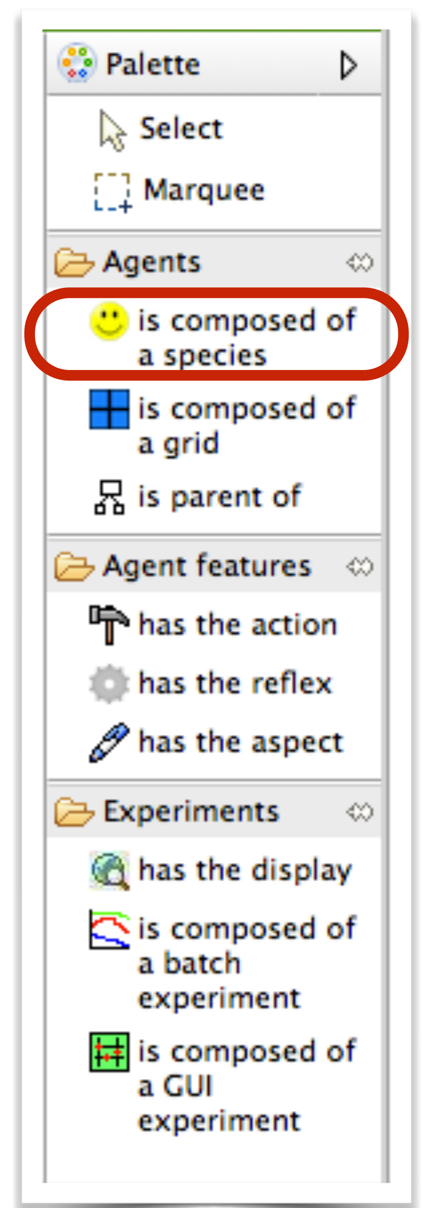
Computation of the world geometry from the envelope of the building shapefile



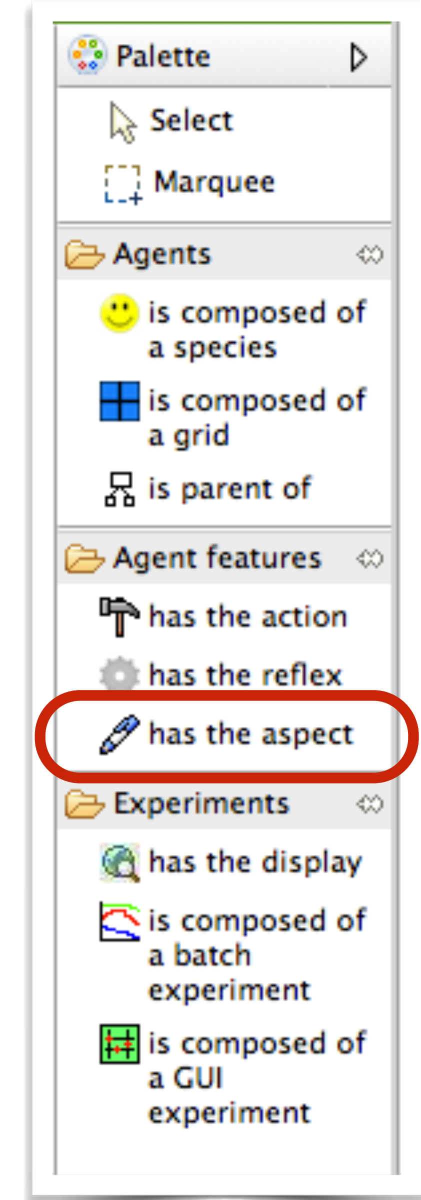


Step 6: people species

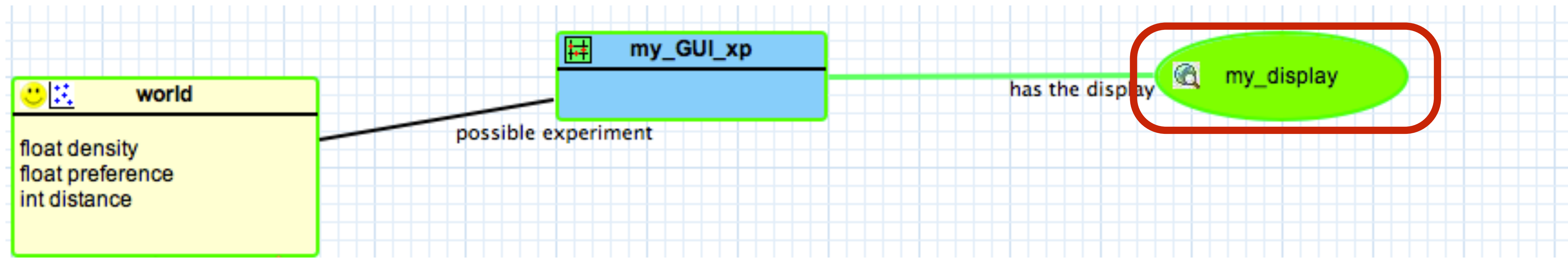
- **Exercice 8**: define inside the world a new species called « people » with 3 variables:
  - my_house: type: house
  - color: type: rgb, init value: probability of 0.5 to be red and yellow
  - is_happy: type: bool



- **Exercice 9**: Add an aspect to the *people* species called « cube »:
  - Add a layer called « People » that draws a *cube* of side size *20* with for color the « *color* » expression (i.e. the color variable of the people agents).



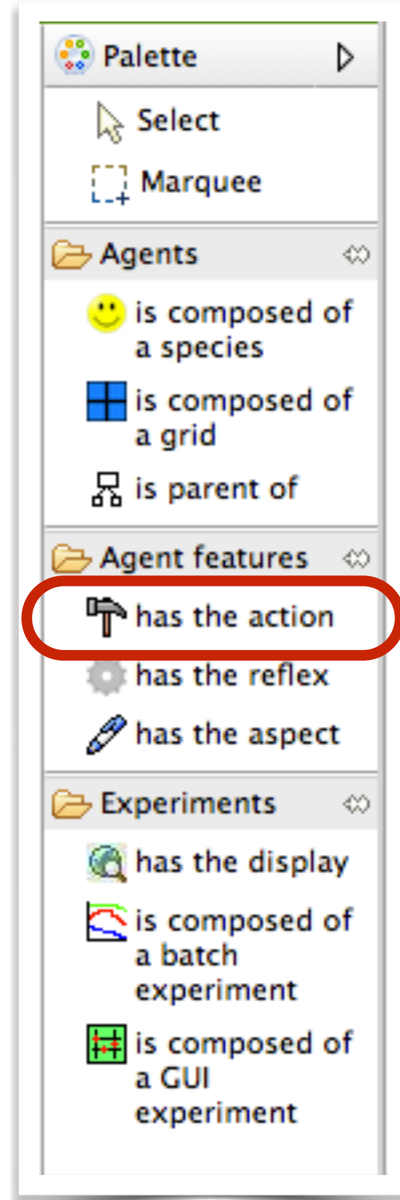
- **Exercice 10**: in the display « my_display » add a new layer called « People », in which the species « people » is displayed with aspect « cube » (the one previously defined)



- **Exercice 11**: Add an action (capability) to the *house* species called « go_out »:

- The action increments by one the capacity of the building (one place more):

```
capacity <- capacity + 1;
```



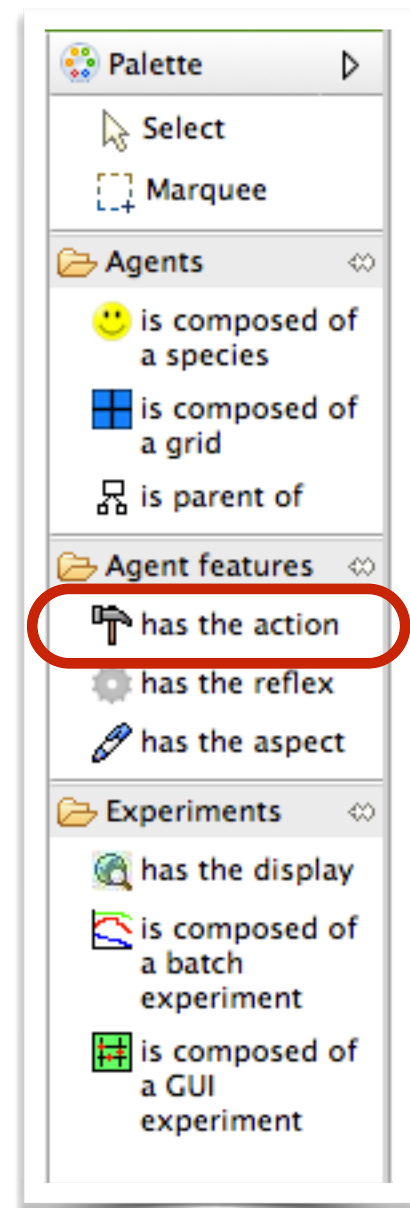
Step 9: go_in house action

- **Exercice 12**: Add an action to the *house* species called « go_in »:
  - Add an argument (input of the action) to this action called « a_people » of type *people*
  - The action decrements by one the capacity of the building (one place less)

```
capacity <- capacity - 1;
```

  - Then the action places the agent inside the building:

```
a_people.location <- any_location_in(shape);
```



The **any_location_in(***an_agent/ a_geometry***)** operator returns a random point inside the geometry/agent

- **Exercice 13**: In the *init* section of the world agent, after creating the *house* agents:
  - compute the total capacity of the *house* agents (sum of *house* capacity):

```
int total_capacity <- sum (house collect each.capacity);
```

  - compute the number of people agents to create (nb of house cells x density) :

```
int nb_people <- total_capacity * density;
```

  - create nb_people *people* cells: for each of them, choose a house which has still the capacity to add this people, then ask the house to apply the *go_in* action with the created people.

```
create people number: nb_people {
   my_house <- one_of (house where (each.capacity > 0)) ;
   ask my_house {
          do go_in a_people:myself;
    }
}
```

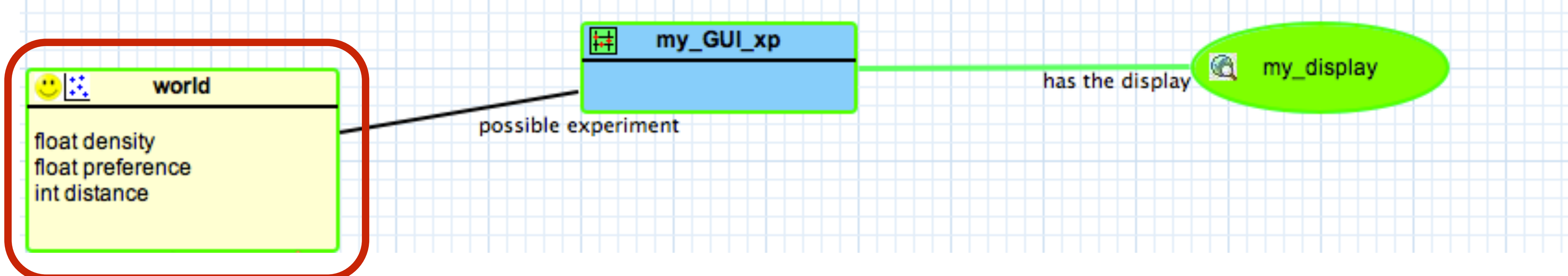
The **sum(*list*)** operator returns the sum of the list

The *list* **collect** *expression* operator returns the list created after applying the expression on each element of the left list

The **ask** statement allows to ask to one or several agents to do something

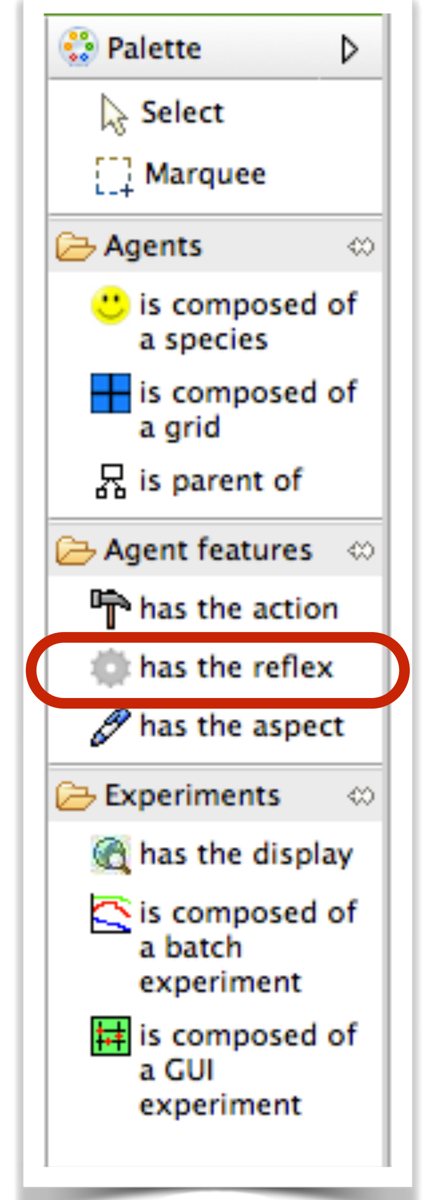
The **do** statement allows to apply an action, the value of argument are given by using the name of the argument + **:**

The **myself** keyword allows to refer to the agent concerned by the previous *context*



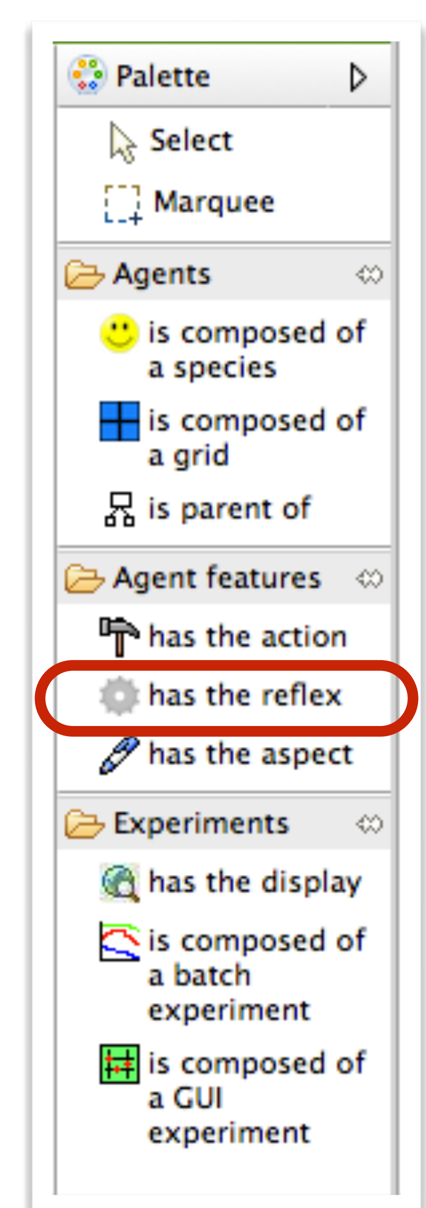
Step 11: people compute happiness reflex

- **Exercice 14**: Add a reflex to the *people* species called « compute_happiness »:
  - compute the list of *people* agents that are located within the distance *distance* to the agent (its neighborhood)
  - compute the number of people agent in the neighborhood
  - compute the number of *people* agents in the neighborhood that have a different color
  - compute the *is_happy* variable: the people agent is happy there is no one in its neighborhood or if the rate of people with a different color is lower than its *preference*



Step 12: people move reflex

- **Exercice 15**: Add a reflex to the *people* species called « move »:
  - add a condition (*Condition*) to the reflex activation: the reflex is activated only if the agent is not happy:
  - Concerning the reflex gamp code, first, ask *my_house* to apply the action *go_out*
  - then, set the variable *my_house* to one house with a capacity higher than 0
  - finally, ask *my_house* to apply the action *go_in* with myself as an argument

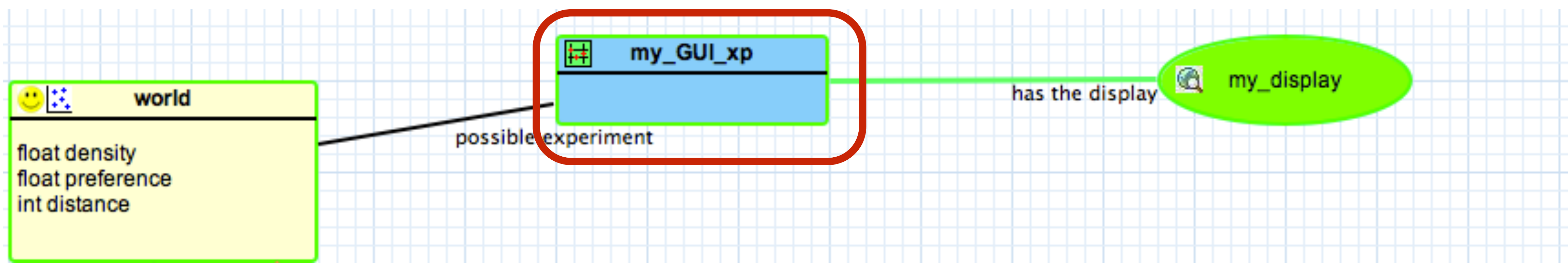


Step 13: monitor

- **Exercice 16**: Add a monitor to the *my_GUI_xp* experiment:
- Text: nb of happy people; value :

```
people count each.is_happy;
```

The *list* **count** *condition* operator returns the number of elements of the list that verifies the right condition

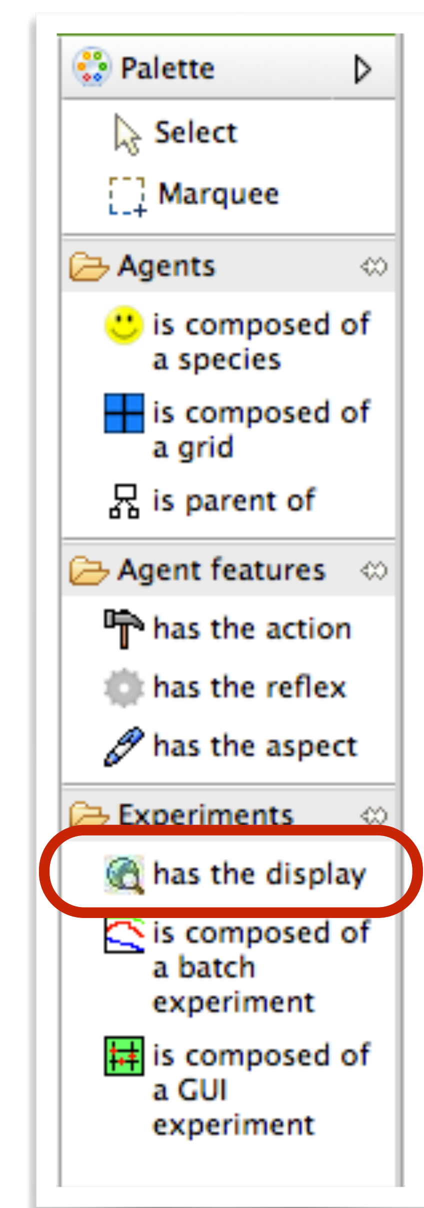


Step 14: chart display

- **Exercice 17**: Add a display to the *my_GUI_xp* experiment called « charts »:

➡ add a new layer called « charts » of type « chart ». This chart is a series chart (nothing to change), with one data series: the number of people that are happy (green color: #green) :

```
people count each.is_happy;
```



Step 15: stop simulation

- **Exercice 18**: Add a reflex to the *world* agent called « end_simulation »:
- This reflex is activated (condition) only when there is no more unhappy people:

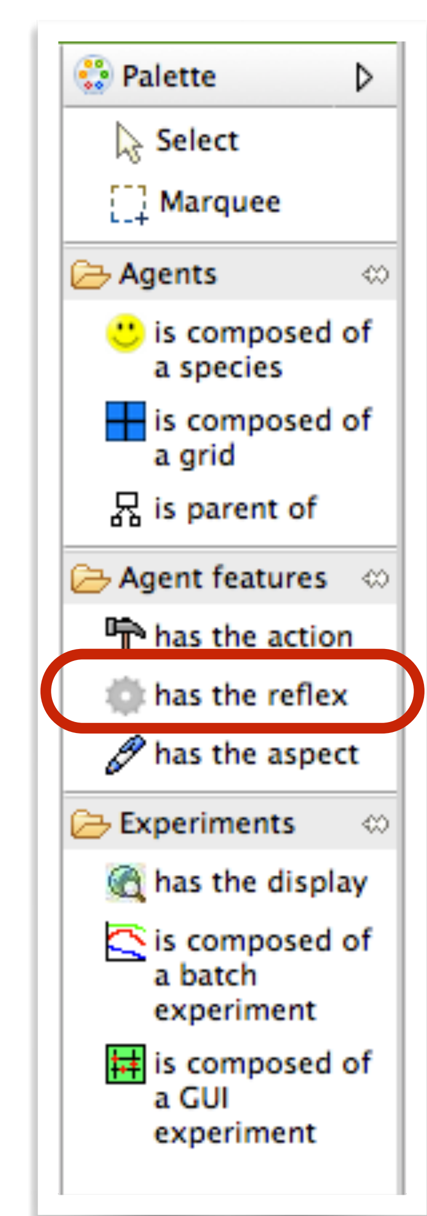
```
empty (people where not each.is_happy)
```

- The reflex pauses the simulation:

```
do pause;
```

pause is an action of the *world* agent

The **empty(*list*)** operator returns *true* when the list is empty (*false* otherwise)



Conclusion of model 2: it is already finished!