From Grid Environment to Geographic Vector Agents, Modeling with the GAMA simulation platform

Patrick Taillandier, Alexis Drogoul

From Grid Environment to Geographic Vector Agents, Modeling with the GAMA simulation platform

Patrick Taillandier[1,2], Alexis Drogoul[1,2]

[1] IRD, UMI UMMISCO 209,
32 avenue Henri Varagnat, 93143 Bondy, France
[2] IFI, MSI, UMI 209,
ngo 42 Ta Quang Buu, Ha Noi, Viet Nam
patrick.taillandier@gmail.com, alexis.drogoul@gmail.com

1. Introduction

These last years has seen an important spreading of the agent-based simulations. If the early models proposed a very basic representation of the spatial environment --usually a grid-- the newest models tend to propose a very precise one that integrates geographic vector data.

Unfortunately, if all of the existing agent-based simulation platforms allow to use a grid as environment, rare of the ones that allow to integrate geographic vector data. Moreover, these platforms are often complex to use and their understanding can require a time investment from the modeler that can be similar to the one needed to develop a model from scratch. Thus, today, most of the models that integrate complex vector data are developed from scratch.

The GAMA agent-based simulation platform [1][2] was developed to address these issues. This platform, which is under GPL license, provides a complete modeling and simulation development environment for building spatially explicit multi-agent simulations. Its main advantages come from its versatility (domain independent) and the simplicity to define a model with it. Indeed, GAMA provides a rich, yet accessible, modeling language, the GAML, that allows to define complex models integrating grids and geographic vector data. In this paper, we present the spatial capabilities of the new version of GAMA (v1.3) and we illustrate its utilization by case study concerning the classic Shelling model [3]. In this case study, we show how to build different kinds of models, from the simplest (grid environment) to the more complex (geographic vector agents) with the GAMA platform.

The paper is organized as follows. In Section 2, we present the GAMA platform, in particular its capabilities concerning the environment representation. Section 3 is dedicated to the presentation of the case study. At last, Section 4 concludes.

2. The GAMA platform

In this section, we briefly introduce some of the major agent-based simulation platforms. Then, we present the GAMA simulation platform, in particular its capacities concerning the environment representation.

2.1. Existing simulation platforms

As mention in Section 1, there are numerous agent-based simulation platforms, however very few allow to directly manipulate a complex environment integrating geographic vector data.

Swarm [4] is a well-established simulation platform and inspiration for many others. Its original version does not allow to integrate geographic vector data. However, a library called Kenge [5] allows to load layers of geographic vector data. Practically, this extension allows to create a cellular automata from a shape file. In addition, an ad hoc access to geographical data has been developed for specific models (e.g. [6]). Unfortunately, they do not provide any spatial primitives neither the possibility to store the resulted environment.

Netlogo [7] is also a well-established simulation platform. The environment is made of a 'patches' grid though agents have their coordinates defined in a continuous space. Patches can have behaviors. Other services offered are: diffusion, distance, agents/patches within a radius, neighbors, etc. The GIS support has been added recently through an extension [8]. It allows import and export of raster and vector data with or without projection. The attributes of the vector data are made accessible as well as their geometrical characteristics (centroid, list of vertex, etc.). Some basic geometrical operations are also available (bounding rectangles, union of polygons, etc.). However, many more advanced spatial analysis operation are not offered.

CORMAS [9] is a platform dedicated to the modeling in ecology and especially the natural resources management where space representation and interaction is essential. It proposes two environment modes: vector and raster. They share the same organization of 3 classes «spatial entity», «agent», and «object». This organization allows to have a standard protocol for movement but make it quite rigid. Within CORMAS, only basic services are provided (like localization, neighborhood, basic movement and perception). The modeler has to implement any other higher-level services.

In 2008, Urbani proposed the SMAG (portmanteau word from SMA-SIG or MAS-GIS in english) architecture linking a GIS and MABS simulator for decision support system. The author implemented it over CORMAS, calling it CORMGIS [10]. The integration is relatively basic as access to geo-referenced data is done through a data-connection to ArcGIS. In addition, no GIS primitive (union, intersection, etc) is available.

Repast J [11] is a modeling toolkit inspired by Swarm. As a toolkit, it provides a structure with only basic services readily available. Different grids are implemented (hexagonal or rectangular, torus or not, etc.) but agents are not (only an interface is given). The GIS support is done through the OpenMap library. It provides the minimal services of a GIS: importing/exporting shapefiles and raster data, some geometrical operations, access to data attributes, etc. Nevertheless, as Repast J provides access to OpenMap[12], the modeler can implement more complex operations. Unfortunately, this programming is far from reach of the vast majority of modelers.

Repast Symphony (Repast S) [13] is the up-to-date version of the Repast toolkit. It provides the same basic features as Repast J, but is not based on the same GIS library but on Geotools [14], and provides additional GIS services. In particular, Repast S allows to directly model a network of lines as a graph and to compute the shortest paths from one point to another. It allows as well to visualize and manage 3D data. Nevertheless, the number of GIS operations available is still fairly limited and localized agents are still to be programmed. More advanced operations have to be programmed (using the Geotools librabry) which is again, far from reach for many modelers.

To conclude, there are numerous agent-based simulation platforms, but these ones are often very limited when working with GIS data (e.g. Swarm, Netlogo, Cormas) or require high level programming skills (Repast S and J).

2.2. The GAMA simulation platform

In order to address these shortcomings, we developed the GAMA platform [1][2], which goes much further by making available many more GIS services and operations and especially an advance management of geographic vector data and provide a complete modeling and simulation development environment. Actually, GAMA provides a rich, yet accessible, modeling language based on XML, the GAML, that allows to define complex models integrating grids and geographic vector data.

The idea of GAMA concerning the management of environment is to locate the agent in a continuous environment, which serves as a reference. In this environment, all of the agents are provided with a geometry. This geometry, which is based on vector representation, can be simple (point, polyline or polygon) or complex (composed of several sub-geometries). The geometry of the agents can be defined by the modeler (a list of points) or directly loaded from a shapefile. Indeed, GAMA allows to use geographic vector data to create agents of a specific species (a prototype of agents that defines both the agent internal state and their behavior): each object of the geographical data will be automatically used to instantiate an agent, GAMA taking care of managing the spatial projection of the data and, if necessary, of reading the values of the attributes. Remark that this kind of geographic data agentification was already used for other application contexts such as cartographic generalization [15]. In the context of simulation, the advantage of this approach is to give the possibility to manage geographic objects exactly like other agents in the simulation: it will be possible to give them an internal state and a behavior. Figure 1 gives an example of the agentification of 4 buildings from a shapefile.

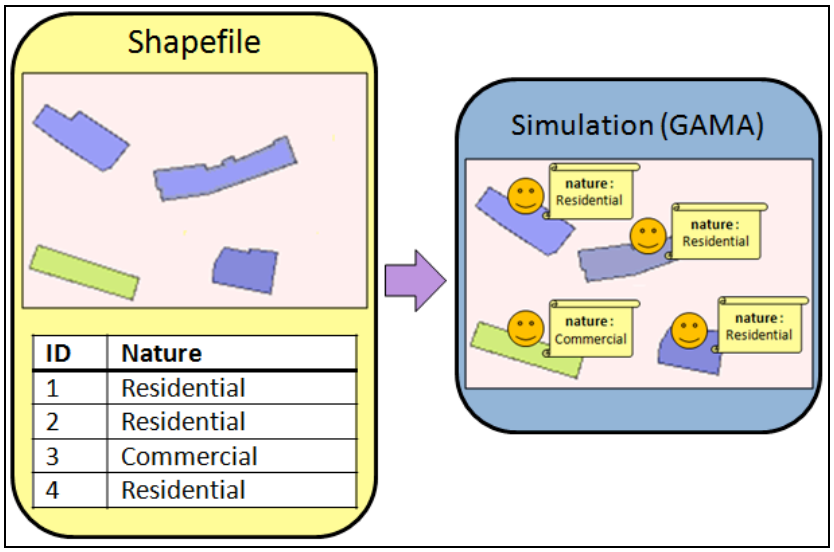

Fig. 1. Example of geographic data agentification

All of the computations related to spatial relationships are made refer to the reference environment. Thus, the distance between two agents, the overlapping of geometries, etc., are directly computed from it. In addition, the role of the reference environment is to adapt all the situated actions of agents (movement, action, interaction) to all registered environments. Thus, for example, when an agent moves, the reference environment manages itself the movement of the agent on the potential grids. This operation is totally transparent for the modeler, who does not have to manage the synchronization of the different environments.

In order to ease the manipulation of the vector geometries, GAMA integrates different GIS features that are directly available through the GAML language. Thus, GAMA allows to:

- Compute the area and the perimeter of a geometry.
- Test if two geometries intersect, touch, cross, overlap each other.
- Compute the convex hull and the buffer geometry of a geometry
- Apply translation, rotation and scaling operations on a geometry
- Compute the geometry resulting from the union, intersection or difference of two geometries.
- Compute the distance between two geometries (minimal distance).
- Compute the neighborhood of an agent, i.e. all the agents that are localized at a distance lower than a given thresholds to the agent.
- Compute a random point inside a geometry.
- Compute the point of a geometry that is the closest to the agent location.
- Apply a tessellation operation (square or triangle) on a geometry (Figure 2).

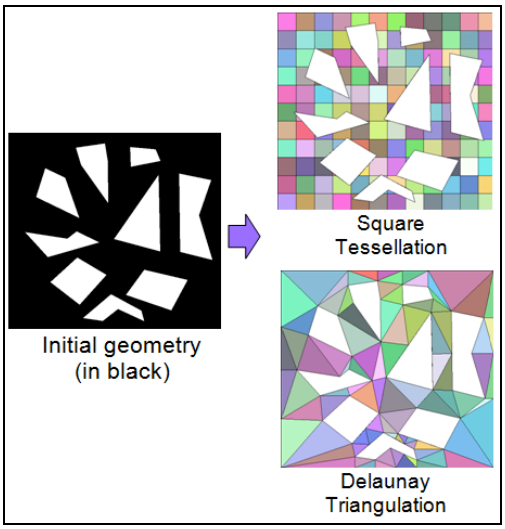

Fig. 2. Example of Tessellations (square and triangle).

- Compute the skeleton of a geometry.
- Compute the shortest path (or the distance) inside a geometry (line network or polygon) between two points located in the geometry. For this computation, our approach consists in modeling the geometry as a graph, and in computing from it the shortest path linking the two points. In the context of a line network, the modeling as a graph is trivial. In the context of a polygon, this one is based on a Delaunay triangulation of the geometry: each triangle resulting from the triangulation is modeled as a node and an edge represents the fact that two triangles are adjacent. Figure 3 shows an example of graph computation. Two algorithms are implemented for the shortest path computation: Dijkstra [16] and Floyd Warshall [17].

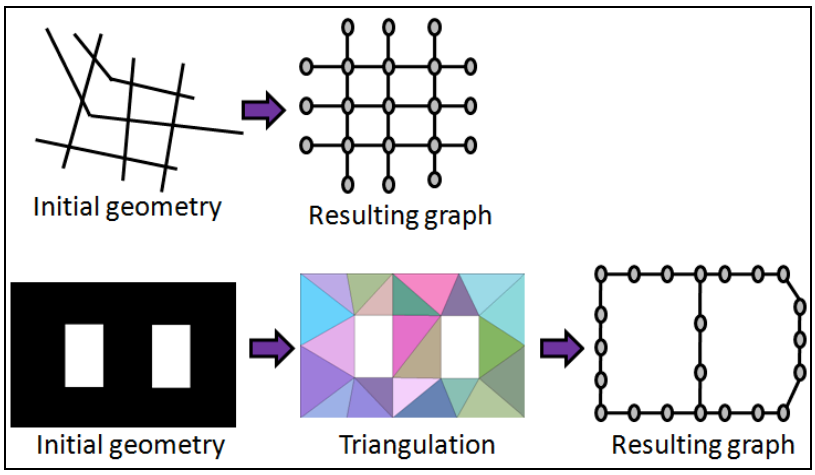

Fig. 3. Example of graph computation

3. Modeling with GAMA: the Schelling Model

In this section, we propose to illustrate how building a model with GAMA by presenting two models derived from the classic Schelling model. The first model, in which the environment is represented by a grid, is close to the original Schelling model. The second model, more realistic, proposes to agentify real geographic vector data.

3.1. The Schelling 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 [3]. 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. This model is a good example of generative model, where the emergence of a phenomenon (here, segregation) is not directly predictable from the knowledge of individual preferences.

In this model, a grid of cells is inhabited by entities of two different colors. Each entity is able to compute the number of neighbors of the same color it has around (i.e. placed the 8 adjacent cells if we consider a Moore neighborhood). Each entity has the same "degree of tolerance" as the others, expressed as a minimum ratio of entities of the same color around. At each time step, if the actual ratio computed is less than its preferred ratio, it moves to another free cell, chosen randomly; otherwise, it stays put.

3.2. Generality on the GAMA models

In GAMA, defining a model means defining four sections:

- *global*: describes the global variables, the parameters, the global dynamics and the initialization of the model
- *entities*: describes the species of agents
- *environment*: describes the environment (the continuous environment and eventually grids)
- *output*: describe the output (displays, files,...) of the model

The skeleton of a GAMA model is defined as follows:

```
<model name="my_model">
   <global>
      <!-- Global variables, parameters and actions -->
   </global>

   <environment>
      <!-- Grid environment(s) in which the agents will be located -->
   </environment>

   <entities>
      <!-- Definition of the species of agents -->
   </entities>

   <output>
      <!-- Definition of the different outputs shown during the simulations -->
   </output>
</model>
```

3.3. Common features

The two models presented in this paper share several common features. Indeed, despite the differences between the two models, they share a great deal of code, as well as algorithms. All these features will be included in the *schelling_common.xml* file which is not a model by itself, but more a "library" for building models of segregation. In this section, we present this file.

3.3.1. Global section

The global section allows to define the parameter of the model: a parameters is a constant global variable that will be accessible to the user through the simulation interface. In this file, we define two parameters:

- *density_of_people*: real
  - Min value=0.01
  - Max value=0.99
  - Default value=0.7
- *ratio_of_similar_wanted*: real
  - Min value=0
  - Max value=1
  - Default value=0.5

In GAML, these two parameters are defined as follows:

```
<var type="float" name="density_of_people" init="0.7" parameter="Density of
  people:" min="0.01" max="0.99" />
<var type="float" name="ratio_similar_wanted" init="0.5" min="0" max="1"
  parameter="Desired ratio of similarity:" />
```

Figure 4 shows the resulting parameter interface.

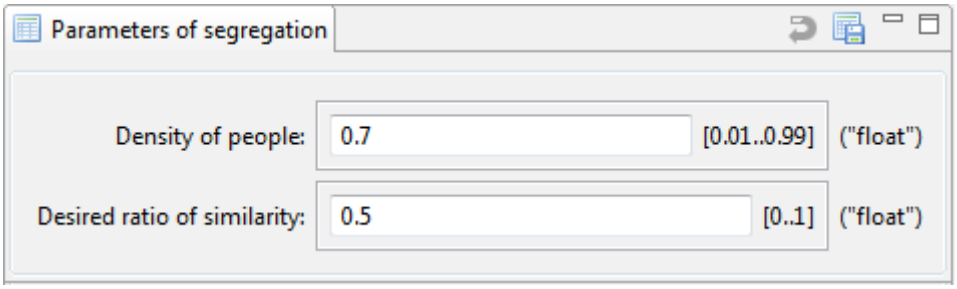

Fig. 4. Resulting parameter interface

In the same way as the parameters, we define four global variables that will be used in the models:

- *number_of_people*: integer
- *all_people*: list of the *people* agents
- *all_places*: list of the places that are available for the *people* agents. In the context of the first model where the environment is represented by a grid, a place will be cell of the grid; in the context of the second model where geographic vector data are used, a place will be a *space* agent (a building from the geographic vector data that has been agentified).
- *sum_happy_people*: number of *people* agents that are happy, computed at each simulation step.

In GAML, These four parameters are defined as follows:

```
<var type="int" name="number_of_people" init="0" />
<var type="list" name="all_people" init="[]" of="people" />
<var type="list" of="space" name="all_places" init="[]" value="shuffle
  all_places"/>
<var type="int" name="sum_happy_people" init="0" value="all_people count
  (each.is_happy)" />
```

Remark that GAML integrates numerous operators concerning the list (e.g. *collect*, *count*, *max_of*, …) that simplify their use.

The last part of the global section concerns the initialization of the model. In the context of our models, this one consists in initializing the places, in computing the number of *people* agents according to the number of places and the density of people chosen and in initializing the *people* agents. As the initialization of the *people* agents and places depend on the model, we choose to define two actions: *initialize_places* and *initialize_people* that will be specifically defined for each model:

```
<init>
   <do action="initialize_places" />
   <set name="number_of_people" value="length all_places * density_of_people" />
   <do action="initialize_people" />
</init>
```

3.3.2. Entities section

As introduced in Section 3.2, another section of a GAMA model consists in defining the entities composing the model. In our context, we define one species of agents: the *people* agents that are situated:

```
<entities>
   <species name="people" skills="situated">
    …
   </species>
</entities>
```

Remark that GAMA offers the possibility to add skills to species of agents. A skill is a set of attributes and primitives that the agents of the considered species will be able to use. For example, the situated skill gives to the agents a geometry, a location, an area, … In the same way it gives the agents the possibility to apply numerous geometric or spatial analysis actions (scaling, rotation, computation of distance between two geometries, …).

Defining a species of agents in GAMA means defining the internal state of the agents (their attributes), their behavior and their visual aspect. In our context, the behavior of the agents will depend on the models, but some of their attributes and their visual aspect will be similar for both models.

We define 6 attributes for the *people* agents:

- size : uses for the visual aspect of the agents
  - Constant, by default = 1
- color : their color (red or yellow). This color marks their belonging to a group.
  - Constant, by default = 'red' or 'yellow'
- my_neighbours: *people* agents that are close to the agent
  - The value of this list, that is computed at each simulation step, depends on each model
- similar_nearby : number of *people* agents of the same group (i.e. of the same color) in the neighborhood
  - Computed at each simulation step
- total nearby : number of *people* agents in the neighborhood
  - Computed at each simulation step
- is_happy: Boolean that indicates if the agent is happy (ratio of *people* agents of the same group high enough).
  - Computed at each simulation step

In GAML, These attributes are defined as follows:

```
<var type="int" name="size" init="1" const="true" />
<var type="rgb" name="color" init="(flip 0.5) ? 'red' : 'yellow'" const="true" />
<var type="list" of="people" name="my_neighbours" init="[]" />
<var type="int" name="similar_nearby" init="0" value=" (my_neighbours count
   (each.color = color)) " />
<var type="int" name="total_nearby" value="length my_neighbours" />
<var type="bool" name="is_happy" value="similar_nearby &gt;=
   (percent_similar_wanted * total_nearby )" />
```

We define a simple visual aspect for the *people* agents consisting in drawing a circle of radius *size* and color *color*:

```
<aspect name="simple">
   <draw shape="circle" size="size" color="color"/>
</aspect>
```

3.3.3. Output section

This section of the model allows to define the outputs of the model, in particular its displays. In our context, the main display (environment and agents) depends on the models. However, we propose to define another display --common for both models-- consisting in a chart representing the evolution of the percentage of happy people.

In GAML, this output is defined as follows:

```
<output>
  <display name="Chart">
    <chart type="series" name="Global happiness and similarity"
      background="rgb 'lightGray'" axes="rgb 'white'">
      <data name="happy" color="rgb 'blue'" value=" (sum_happy_people /
        number_of_people) * 100" style="spline" />
    </chart>
  </display>
</output>
```

3.4. Model based on a grid environment

The first model proposed is very close to the original Schelling model. In this model, the environment is represented by a grid on which each *people* agent is localized. Figure 5 shows the model implemented.

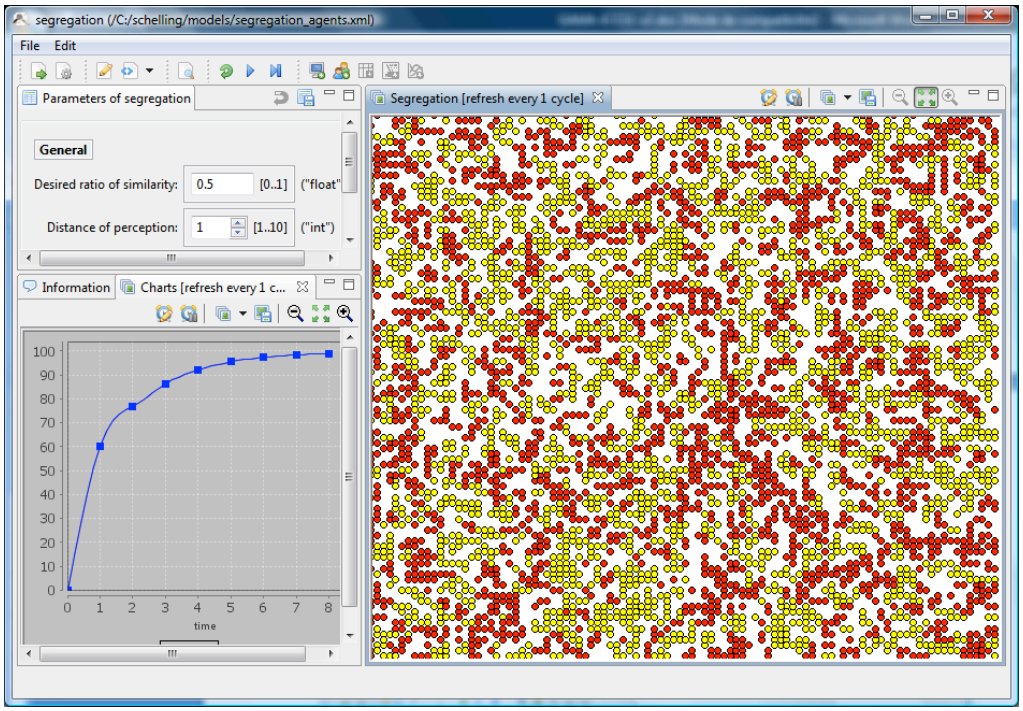

Fig. 5. Implementation of the model based on a grid environment

As mentioned in Section 3.3, we use the *schelling_common.xml* file as a base. This possibility is enabled by the use of the *include* instruction, which allows a GAML model to "load" definitions located in other files. The *include* instruction basically does a copy of the included definitions, but also allows to override attributes, actions and even behaviors defined in the *global* section. Thus, our model is defined as follows:

```
<model name="segregation_grid">
  <include file="../include/schelling_common.xml"/>
  …
</model>
```

3.4.1. Global section

Concerning the global section, we define one new parameter:

- Distance of perception: given by a number of cells

- Min = 1
- Max = 10
- By default= 1

This attribute is defined by:

```
<var type="int" name="neighbours_distance" init="1" max="10" min="1"
   parameter="Distance of perception:" category="Population" />
```

We define as well a new global variable:

- Free places: places that are not occupied by a people agent.

This global variable is defined by:

```
<var type="list" name="free_places" init="[]" of="space"/>
```

At last, we define the two initialization actions mentioned in Section 3.3.1:

- Initialize_places: initialize the *all_places* and *free_places* lists:

```
<action name="initialize_places">
  <set name="all_places" value="shuffle (space as list)" />
  <set name="free_places" value="all_places"/>
</action>
```

- Initialize_people: create the *people* agents and initialize the *all_people* list:

```
<action name="initialize_people">
  <create species="people" number="number_of_people" />
  <set name="all_people" value="list people"/>
</action>
```

3.4.2. Environment section

The next section we have to define concerns the environment. Indeed, in this model, the environment is represented by a grid. The grid have a size of 100 × 100. The number of agent in a cell is limited to one. Concerning the computation of the neighborhood, we propose to use the Moore neighborhood. Such a grid is built by the following GAML lines:

```
<environment>
  <grid name="space" width="100" height="100" neighbours="8" torus="false" >
    <var type="bool" name="multiagent" value="false" const="true" />
  </grid>
</environment>
```

3.4.3. Entities section

Concerning the *people* agents, we define a new attribute:

- *my_neighbours*: list of the *people* agents located in distance lower than *neighbours_distance* (a number of cells) to the agent:

```
<var type="list" name="my_neighbours" value="self neighbours_at
  neighbours_distance" of="people" />
```

In addition, we define an initialization procedure for the *people* agent: when a *people* agent is created, the cell on which the agent is located (randomly chosen) is removed from the *free_places* list:

```
<init>
      <remove item="space location" from="free_places"/>
</init>
```

We have to define a behavior for the *people* agents. This one is very simple: when a *people* agent is not happy, it moves to one of the free places, and then updates the free place list. In GAMA, the simplest way of defining such behaviour consists in defining a *reflex*. A *reflex* is an action that is triggered when a condition is checked. In our context, the *reflex* is triggered when the agent is not be happy:

```
<reflex name="migrate" when="!is_happy">
      <add item="space location" to="free_places"/>
      <set name="location" value="any free_places" />
      <remove item="space location" from="free_places"/>
</reflex>
```

3.4.4. Output section

Concerning the output section, we define the main display: in this one, the *people* agents are displayed using their *simple* aspect (the one defined in the *schelling_common.xml* file):

```
<output>
    <display name="Segregation" >
       <species name="people" aspect="simple"/>
    </display>
</output>
```

3.5. Model based on geographic vector agents

The first model showed how building the classic Schelling model. In this section, we propose to modify this model in order to make it more realistic. Indeed, as shown in [18], passing from a grid representation of the environment to a vector one has deep impact on the result and the realism of the model. In this context, we show that GAMA allows to easily define a model integrating geographic vector data. Actually, passing from the classic Schelling model to a vector one requires very few changes.

This model is similar to the classic Schelling model, but the entities do not anymore inhabit a grid but a set of buildings. The neighborhood of the entities is directly computed from the real distance between the different entities. Each building has a max capacity of entities that represents the maximum number of entities that can stay in the building at the same time. Figure 6 shows the model implemented.

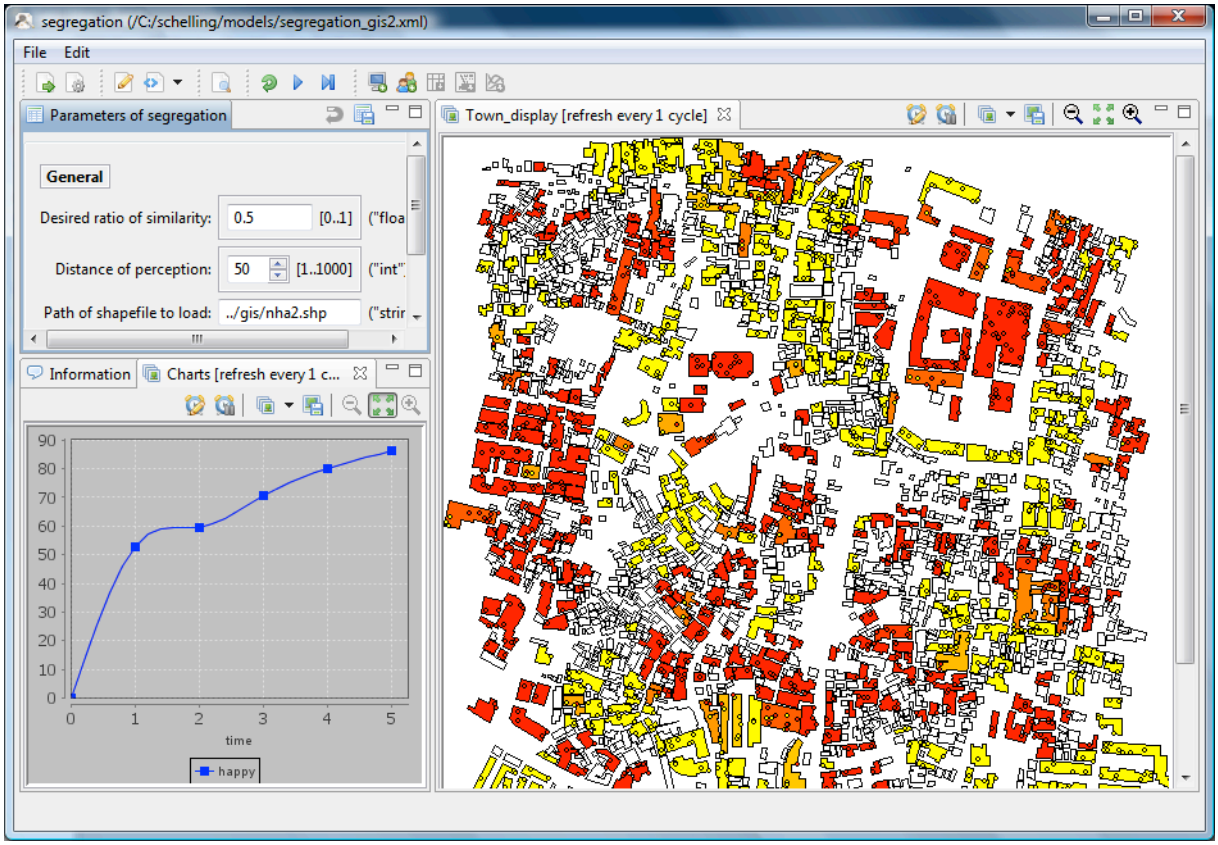

Fig. 6. Implementation of the model based on geographic vector agents

3.5.1. Global section

Concerning the global section, we define two new parameters:

- Distance of perception: given by a distance in meters
  - Min =1
  - Max = 1000
  - By default= 50
- Path of the shapefile for the buildings

```
<var type="int" name="neighbours_distance" init="50" min="1"
  parameter="Distance of perception:" category="Population" max="1000"/>
<var type="string" name="shape_file_name" init="'building.shp'"
  parameter="Path of shapefile to load:" category="GIS specific" />
```

In addition, we define the two initialization actions mentioned in Section 3.3.1:

- *Initialize_places*: agentification of the building shapefile – one *space* agent is created per building in the shapefile-- and initialization of the *all_places* list; in this model, a place is space agent, i.e. a building. We use the CAPACITY attribute of the shapefile to initialize the *capacity* attribute of the *space* agents. In GAML, this action is defined as follows:

```
<action name="initialize_places">
   <create species="space" from="shape_file_name" with="[capacity :: read
      'CAPACITY']" return="spaces" />
   <set name="all_places" value="shuffle spaces" />
</action>
```

- *Initialize_people*: this action creates the *people* agents and initializes the *all_people* list. The number of *people* agents is computed according to the total capacity of the building (i.e. the sum of the building capacity) and the density_of_people parameter. For each *people* agent created, the action *move_to_new_place* is called. This action is described in Section 3.5.3. In GAML, the *Initialize_people* action is defined as follows:

```
<action name="initialize_people">
   <set name="number_of_people" value="density_of_people * sum (all_places collect
      ((space each).capacity))" />
   <create species="people" number="number_of_people" return="created_people" />
   <ask target="created_people">
      <do action="move_to_new_place" />
   </ask>
   <set name="all_people" value="list people" />
</action>
```

3.5.2. Environment section

The next section that we have to define concerns the environment. In this model where no grid is used, we just have to specify the size of the environment. GAMA allows to directly compute the size of the environment from a shapefile. In our model, we propose to use the building shapefile to compute the size of the environment:

```
<environment  bounds="shape_file_name"/>
```

3.5.3. Entities section

People agents

For the people agent, we have to define several new attributes:

- *red*: value of the red component of the agent color (RGB : [0, 255])
- *green*: value of the green component of the agent color
- *blue*: value of the blue component of the agent color
- *current_building* : building on which the agent is localized
- *my_neighbours*: list of the *people* agents located in distance lower than *neighbours_distance* (a distance in meters) to the agent.

```
<var type="int" name="size" init="5" const="true" />
<var type="int" name="red" init="list color at 0" const="true" />
<var type="int" name="green" init="list color at 1" const="true" />
<var type="int" name="blue" init="list color at 2" const="true" />
<var type="space" name="current_building" init="nil" />
<var type="list" name="my_neighbours" value="(self neighbours_at
   neighbours_distance) of_species people" />
```

We have to define the *move_to_new_place* action. This action moves the agent to the first *space* agent (i.e. building) of which the capacity is higher than 0. Then, the *people* agent asks the selected *space* agent to trigger its *accept* action. This action is described below. The *move_to_new_place* action is defined as follows:

```
<action name="move_to_new_place">
  <set name="current_building" value="(all_places first_with
    ((each.capacity) > 0))"/>
  <ask target="current_building">
    <do action="accept">
      <arg name="one_people" value="myself" />
    </do>
  </ask>
</action>
```

We have to define the behavior of the *people* agents. This one is simple: when an agent is not happy, it asks the *space* agent on which it is located to trigger its *remove* action, then it triggers its *move_to_new_place* action:

```
  <reflex name="migrate" when="!is_happy">
    <ask target="current_building">
      <do action="remove">
        <arg name="one_people" value="myself" />
      </do>
    </ask>
    <do action="move_to_new_place" />
  </reflex>
```

Space agents

As mentioned in Section 3.3.1, in this model, the buildings are modeled as *space* agents. Thus, we have to define this species of agents. The *space* agent are situated: we give them a *situated* skill:

```
<species name="space" skills="situated">
 …
</species>
```

These agents have two attributes:

- the list of *people* agents that are localized on the agent
- the capacity of the building. We remind that this attribute is directly initialized by reading the CAPACITY attribute of the shapefile (Section 3.5.1)

These two attributes are defined as follows:

```
<var type="list" name="agents" of="people" init="[]" />
<var type="int" name="capacity" />
```

As mentioned in the previous sections, we have to define two actions:

- *accept*: takes a *people* agent as parameter. This action adds this *people* agent to the *agents* list, then it places the *people* agent at a random location on the agent geometry, and finally, it updates the agent capacity:

```
<action name="accept">
  <arg name="one_people" />
  <add item="one_people" to="the agents of self"/>
  <set name="location of (one_people as people)" value="self
    place_in[agent::self]" />
  <set name="capacity" value="capacity - 1" />
</action>
```

- *remove*: takes a people agent as parameter. This action removes this *people* agent from the *agents* list, then it updates the agent capacity:

```
<action name="remove">
   <arg name="one_people" />
   <remove item="one_people" from="the agents of self"/>
   <set name="capacity" value="capacity + 1" />
</action>
```

We define as well an aspect for the *space* agents consisting in drawing the geometry of the agent (a polygon) with a color that depends on the *people* agents on the building: the color of the *space* agent is the mean of the color of the *people* agents:

```
<aspect name="gis">
   <draw shape="geometry" color="empty agents ? 'white' : [mean (self.agents collect
      each.red), mean (self.agents collect each.green), mean (self.agents collect
      each.blue)]" />
</aspect>
```

3.5.4. Output

Concerning the output section, we define a new display: in this one, the *people* and *space* agents are displayed:

```
<display name="Town_display" refresh_every="1">
   <species name="space" aspect="gis" />
   <species name="people" aspect="simple"/>
</display>
```

4. Conclusion

In this paper, we present the last version of the GAMA platform (version 1.3) [2]. We illustrated the capacities of this platform by a case study concerning the Schelling model. In particular, we showed that GAMA allows by just changing few lines of code to pass from a grid representation of the environment to geographic vector agents.

This version of GAMA is already used in several projects related to different application domains such as the avian flu local propagation in North Vietnam [19], the rift valley fever in Senegal, the brown hopper invasion in South Vietnam [20], the effect of emotions on waves of panic [21] and emergency response in Hanoi [22].

The next version of GAMA, version 1.4, is going to include a new integrated development environment (IDE) with a new modeling language. The goal is to ease the work of the modelers by providing a less extensive and easier to learn language. This version will also include all the classic features provide by most of the modern IDE (auto-completion, automatic detection of errors, etc.).

References

1. Amouroux, E., Chu, T. Q., Boucher, A., Drogoul, A.: GAMA: An Environment for Implementing and Running Spatially Explicit Multi-agent Simulations. *PRIMA07*, 359-371 (2007)
2. GAMA platform, http://gama-platform.googlecode.com (September 2010).
3. Schelling, T.C.: Dynamic models of segregation. Journal of Mathematical Sociology, 1 (1), 143–186 (1971).
4. Minar, N., Burkhart , R., Langton, C., Askenazi M.: The Swarm Simulation System: A Toolkit for Building Multi-Agent Simulations, *SFI Working Paper* 96-06-042 (1996)

5. Box, P.: Spatial Units as Agents. *Integrating GIS and Agent-Based Modelling Techniques*. ed. Oxford (2002)
6. Haklay, M., O'Sullivan, D., Thurstain-Goodwin, M. Schelhorn, T.: So Go Downtown": Simulating Pedestrian Movement in Town Centres, *Environment and Planning B: Planning and Design*, 28(3): 343-359 (2001)
7. Wilensky, U. NetLogo. http://ccl.northwestern.edu/netlogo/. *Center for Connected Learning and Computer-Based Modeling*, Northwestern University. Evanston, IL (1999)
8. Russell, E., Wilensky, U.: Consuming spatial data in NetLogo using the GIS Extension. *The annual meeting of the Swarm Development Group*. Chicago, IL (2008)
9. Bousquet, F., Bakam, I., Proton, H. Le Page, C.: Cormas: common-pool resources and multi-agents systems. *IEA/AIE* (Vol. 2) 826-837 (1998)
10. Urbani, D., Delhom, M.: Analyzing Knowledge Exchanges in Hybrid MAS GIS Decision Support Systems, Toward a New DSS Architecture, *LNCS 4953*, 323-332 (2008)
11. North, M. J., Collier, N. T., Vos, J. R.: Experiences Creating Three Implementations of the Repast Agent Modeling Toolkit. *ACM Transactions on Modeling and Computer Simulation*, Vol. 16, Issue 1, 1-25 (2006)
12. OpenMap, http://www.openmap.org/ (October 2010)
13. North, M. J., Tatara, E., Collier, N. T., Ozik, J.: Visual Agent-based Model Development with Repast Simphony, *Conference on Complex Interaction and Social Emergence* (2007)
14. Geotools, http://www.geotools.org/ (October 2010)
15. Ruas, A., Duchêne, C:. A prototype generalisation system based on the multi-agent system paradigm. *Generalisation of Geographic information: cartographic modelling and applications*, Elsevier Ltd, 269-284 (2007)
16. Dijkstra, E. W.: A short introduction to the art of programming. *Technological Univ. Eindhoven*, Rep. EWD316 (1971)
17. Floyd, R.W.: Algorithm 97: Shortest Path, *Communications of the ACM*, vol. 5, n^{o} 6, p. 345 (1962)
18. Crooks, A.T.: Constructing and implementing an agent-based model of residential segregation through vector GIS. *IJGIS.* Vol. 24, No. 5, 661–675 (2010)
19. Amouroux, E., Desvaux, S., Drogoul, A.: Towards Virtual Epidemiology: An Agent-Based Approach to the Modeling of H5N1 Propagation and Persistence in North-Vietnam. *PRIMA2008:* 26-33 (2008)
20. Phan, C.H., Huynh, H.X., Drogoul A.: An agent-based approach to the simulation of Brown Plant Hopper (BPH) invasions in the Mekong Delta (L). *RIVF* (2010).
21. Le, V.M., Adam C., Gaudou B., Ho, T.V., Taillandier, P.: Simulation of emotion dynamics in a group of agents in an evacuation situation. *PRACSYS* (2010).
22. Chu, T. Q., Drogoul, A., Boucher, A., Zucker J.D.: Interactive Learning of Independent Experts' Criteria for Rescue Simulations. *Journal of Universal Computer Science,* vol. 15, no. 13, 2701-2725 (2009)