GAMA : bringing GIS and multi-level capabilities to multi-agent simulation

Patrick Taillandier, Duc-An Vo, Edouard Amouroux, Alexis Drogoul

GAMA: bringing GIS and multi-level capabilities to multi-agent simulation

Patrick Taillandier[1,2], Vo Duc An[1,2], Edouard Amouroux[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, vdan@ifi.edu.vn, edouard.amouroux@ird.fr,
alexis.drogoul@gmail.com

Abstract. The agent-based modeling is now widely used to study complex systems. Its ability to represent several levels of interaction along a detailed (complex) environment representation favored such a development. However, in many models, these capabilities are not fully used. Indeed, only simple, usually discrete, environment representation and one level of interaction (rarely two or three) are considered in most of the agent-based models. The major reason behind this fact is the lack of simulation platforms assisting the work of modelers in these domains. To tackle this problem, we developed a new simulation platform, GAMA. This platform allows modelers to define spatially explicit and multi-level models. In particular, it integrates powerful tools coming from Geographic Information Systems (GIS) and Data Mining easing the modeling and analysis efforts. In this paper, we present how this platform addresses these issues and how such tools are available right out of the box to modelers.

Keywords: Agent-based modeling, Geographic vector data, Multi-level models

1 Introduction

The agent-based modeling has brought a new way to study the complex systems. It allows to take into account different levels of interactions as well as the heterogeneity of the entities composing the system.

Even if numerous simulation platforms exist, most of the complex models are still developed from scratch. Indeed, very few platforms allow to directly work with geographic vector data (series of coordinates defining geometries) and/or to define multi-level models. 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.

In this paper, we present the last version of the GAMA multi-agent simulation platform [1][2] and in particular the new capabilities concerning the integration of

GIS data and the development of multi-level models. GAMA provides a complete modeling and simulation development environment for building spatially explicit multi-agent simulations. Many models have already been implemented using this platform (e.g. [3][4][5][6]). 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 based on XML, GAML, that allows to define complex models integrating at the same time entities of different scales and geographic vector data.

The paper is organized as follow. In Section 2, we present the capabilities of GAMA concerning the integration of geographic vector data. Section 3 is dedicated to the presentation of its multi-scale modeling capabilities. Section 4 presents two actual models developed with GAMA. Section 5 discusses about the contributions of the paper. At last, Section 6 concludes.

2 Integrating geographic vector data in simulation

2.1 Use of geographic vector data in simulation

These last years have seen the development on a large scale of geographic vector datasets. Today, most of the decision makers use this type of data when they have to face a problem integrating a spatial dimension. In the context of simulations, using this type of data allows to make the simulations closer to the field situation. Indeed, as shown in [7], 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 addition, it allows to use tools, like spatial analysis, coming from Geographic Information Systems (GIS) to manage these data.

If more and more models integrate geographic vector data, their use can take different forms from the simplest (reading/writing of geographic data) to the most complex (agentification of geographic data).

The most basic functions concerning the use of geographic vector data are the reading and the writing of geographic data from files and from database. The goal is to integrate seamlessly the vector data as the simulation's environment (input) and to store the resulting environment (output). Once geographic vector data has been read, several uses can be made of them. The most straightforward one consists in translating them as a grid where agents are localized.

A more complex use consists in using these data as a "background layer" constituted of geographic objects: the agents will be able to move according to this layer. For example, some agents will be able to move along a network of road, or inside a complex polygon (e.g. inside a forest represented by a polygon. This use requires the integration in the simulator of GIS specific primitives such as moving an agent inside a geometry, computing a shortest path between two points of this geometry (or on a network), etc.

A richer ways of integrating geographic vector data in a model is to consider each geographic object as an agent. Thus, a road will be an agent, a building or a city, and each object contained in a geographic dataset will also be represented by an agent.

Remark that this kind of geographic data agentification was already used for other application contexts such as cartographic generalization [8]. 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. Reciprocally, it is possible to go further and to consider that every "spatialized" (localized and with a geometry) agents of the simulation has a geometry and can be viewed as a geographic object in a geographic dataset. In this way, the management of agents and geographic objects is equivalent and trouble-free. Indeed, no difference is made anymore between agents and geographic objects.

2.2 Geographic vector data in existing simulation platforms

Swarm [9] 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 [10] allows to load layers of geographic vector data. Practically, this extension allows to create a cellular automata from a shapefile. In addition, an ad hoc access to geographic data has been developed for specific models (e.g. [11]). Unfortunately, they do not provide any spatial primitives neither the possibility to store the resulted environment.

Netlogo [12] is also a well-established simulation platform. It is largely used for educational purpose and for research. The GIS support has been added recently through an extension [13]. It allows import and export of vector data and support the projection system (the method used to represent the geographic data on a plane). 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 [14] 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, though being rigid, ease the development of model by abstracting the interaction with environment, thus allows to switch from a discrete environment to a continuous (or vector) one. Unfortunately, CORMAS provides only basic services for the discrete environment. Moreover, GIS support is limited to loading and storing shapefiles (a popular vector data format) and creating elementary areas. GIS primitives (union, intersection, shortest path, etc.) and access to polygon attributes have to be programmed. 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 [15]. 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 [16] 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, the modeler can implement more complex operations. Unfortunately, this programming is far from reach of the vast majority of modelers.

Repast Symphony (Repast S) [17] is the up-to-date version of the Repast toolkit. It provides the same basic features as Repast J, but is based on a more advanced GIS library, Geotools, which 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 are 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, evidently, far from reach for many modelers.

2.3 Geographic vector data in GAMA

In order to address these shortcomings we developed the GAMA platform, which goes much further by making available many more GIS services and operations and especially an advance management of geographic vector data.

The first version of GAMA that was presented in [1] proposed the idea of using a continuous environment to serve as a reference for all other environments (e.g. grid environment). In this former version, all situated agents had a point for geometry. The use of geographic vector data was very limited: there were just to initialize the initial location of the agents and as a background layer.

If the new version of GAMA (GAMA 1.3) kept the same idea of a reference environment, it goes further by providing a true geometry to all situated agents. 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 geographic 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. Consequently GAMA considers localized agents and geographic objects in the exact same way.

Example: the following GAML lines allow to create a set of building agents from the shapefile *shape_file_building.shp* and to set the value of the attribute nature of each created building agent according to the attribute *NATURE* of the shapefile:

```
<create species="building" from="shape_file_building.shp"
  with="[nature:: read 'NATURE']"/>
```

Figure 1 gives an example of the agentification of 4 buildings from a shapefile.

In the same way, GAMA allows to save a set of agents in a shapefile.

Example: the following GAML lines allow to save all the agents of the species *building* in the shapefile *shape_file_building.shp* and to set the value of the attribute *NATURE* of each geographic object according to the attribute *nature* of the agents:

```
<save species="building" to="shape_file_building.shp"
 with="[nature:: 'NATURE']"/>
```

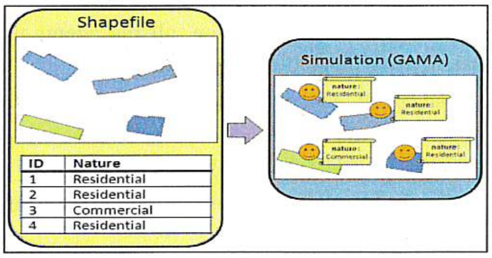

Fig. 1. Example of geographic data agentification

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.

Example: The following GAML line allows to compute the area of the geometry of the agent *ag*:

```
<let name="the_area" value="ag.area" />
```

- Test if two geometries intersect, touch, cross, overlap each other.

Example: The following GAML lines allow to test if the geometry of the agent that is applying the action intersects the geometry *geom*:

```
<do action="interection" return="is_true">
   <arg name="geometry" value="geom" />
</do>
```

- Compute the convex hull and the buffer geometry of a geometry.

Example: The following GAML line allows to compute the convex hull of the geometry of the agent that is applying the action:

```
<do action="convex_hull" return="result"/>
```

- Apply translation, rotation and scaling operations on a geometry.

Example: The following GAML lines allow to rotate the geometry of the agent that is applying the action with an angle of 90°:

```
<do action="rotation ">
   <arg  name="angle" value="90" />
</do>
```

- Compute the geometry resulting from the union, intersection or difference of two geometries.

Example: The following GAML lines allow to compute the difference between the geometry $geom_1$ and the geometry $geom_2$:

```
<do action="difference" return="result">
   <arg name="geometry1" value="geom1" />
   <arg name="geometry2" value="geom2" />
</do>
```

- Compute the distance between two geometries (minimal distance).

Example: The following GAML lines allow to compute the distance between the geometry of the agent that is applying the action and the geometry *geom*:

```
<do action="distance_geometry" return="result">
   <arg name="geometry" value="geom" />
</do>
```

- 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.

Example: The following GAML lines allow to compute the neighborhood of the agent *ag*:

```
<let name="neighborhood" value="ag.neighbours_geometry "/>
```

- Compute a random point inside a geometry.

Example: The following GAML lines allow to compute a random point inside the geometry *geom*:

```
<do action="place_in" return="result">
   <arg name="geometry" value="geom" />
</do>
```

- Compute the point of a geometry that is the closest to the agent location.

Example: The following GAML lines allow to compute the point of the geometry *geom* that is the closest to the agent that is applying the action:

```
<do action="closest_point_in" return="result">
   <arg name="geometry" value="geom" />
</do>
```

- Apply a tessellation operation (square or triangle) on a geometry (Figure 2).

Example: The following GAML lines allow to compute the Delaunay triangulation of the geometry (polygon) *geom*:

```
<do action="triangulation" return="result">
   <arg name="geometry" value="geom" />
</do>
```

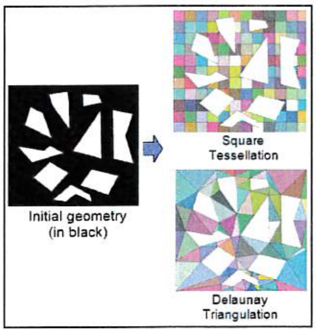

Fig. 2. Example of Tessellations (square and triangle).

- Compute the skeleton of a geometry.

Example: The following GAML lines allow to compute the skeleton of the geometry (polygon) *geom*:

```
<do action="skeletonization" return="result">
   <arg name="geometry" value="geom" />
</do>
```

- 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 4 shows an example of graph computation. Two algorithms are implemented for the shortest path computation: Dijkstra [18] and Floyd Warshall [19].

Example: the following GAML lines allow to move the agent that is applying the action toward the point *the_target*, at a speed of 5 km/h, inside the geometry *geom (which can be a graph or a polygon)*:

```
<do action="goto">
   <arg name="target" value="the_target" />
   <arg name="speed" value="5 km/s" />
   <arg name="geometry" value="geom" />
</do>
```

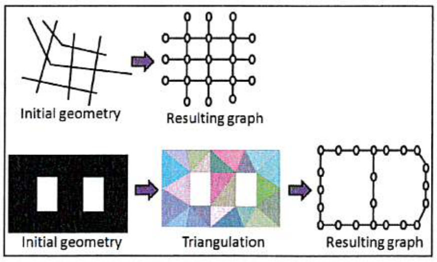

Fig. 3. Example of graph computation

3 Multi-scale Modeling

3.1 Context

Another advantage of the agent-based modeling approach is its representation versatility. Indeed, an "agent" can represent any individual or aggregation/structure of individuals of the reference system, at any spatial scale and across different time horizons. Thus the modeler is free in her/his choice of the entities of the reference system that will be represented by agents. This choice will depend on the level of abstraction of the reference system the modeler is working with. This, in turn, depends on the question he/she wants to answer with the model, on the data available at hand, on the scale at which this data is described, etc.

In addition to the agent representing entities of the reference system, the modeler can need to explicitly represent emergent structures. Indeed, during the simulation stage (execution of the model), some structure can emerge: appearance of pheromone trail built by ant [20], evolution of social group within a population [21], formation of arches in granular environment [22], etc. These structures are often the result of non-linear interactions between the agents defined in the model and can play a significant role in the model dynamics. They can be considered as a higher level of abstraction (upper scale) compare to the underlying agents composing them. It is important, if not crucial, to be able to detect and to generate them dynamically (i.e. might simplified the simulation run).

Current agent-based modeling platforms lack support in term of agent-based modeling language to represent these structures as explicit entities in the model and tools to detect them. Thus, modelers face difficulties when they need to represent them and to follow their dynamics during the course of the simulation.

3.2 Multi-scale modeling in GAMA

In GAMA, in order to let modelers dynamically track the emergence of dynamic structures, we let them represent these structures as explicit entities in the model. We call these entities "emergent agents". As regular agent, an emergent agent can have attributes and behaviors. Beside, its instantiation depends on the appearance of certain properties during the simulation and its life-cycle possesses some specific operations.

3.2.1 Representing emergent structure

The "creation" operation helps to specify when an emergent agent is instantiated. This operation allows the modeler to express in an explicit way the rules governing the instantiation of emergent agents during the simulation. For example, consider a simulation of city dynamics: a modeler can decide to instantiate an emergent agent of species *building block* when two or more *building* agents are close enough. Figure 5 illustrates this example: an emergent agent *(building block)* representing the emergent structure is created with six micro-agents *(building)* as components.

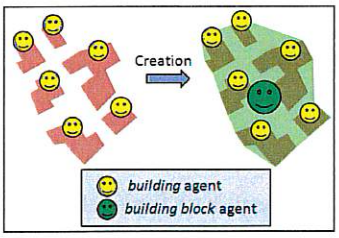

Fig. 4. Creation of an emergent agent (*building block* agent)

The "update" operation describes how micro-agents are added to or removed from an emergent agent. Some micro-agents may no longer satisfy a condition to belong to an emergent agent, while others, still "free" may now fulfill it: this operation allows to specify how these agents are added or removed from the structure. The purpose of this operation is to keep the list of components up-to-date with respect to the meaning of the emergent agent.

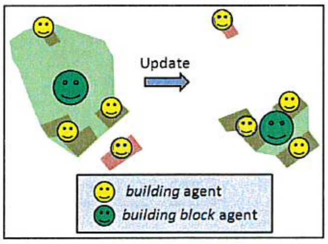

Fig. 5. Update of an emergent agent (*building block* agent)

Figure 6 illustrates the "update" operation. It follows the example of city dynamic simulation presented Figure 5. We consider that a building block agent composes of three *building* agents. One building agent doesn't satisfy the condition to belong to the building block agent anymore. A free building agent satisfies the condition to become a member of the building block agent. This operation helps the modeler to remove one building agent from the building block agent and add one building agent to the building block agent.

The "merge" operation allows the modeler to specify how several emergent agents representing different structures can be merged into one unique emergent agent. The fusion of their respective components then becomes the components of the new unique emergent agent.

Figure 7 illustrates the "merge" operation using the same example as Figure 5 and 6. We consider a new *building block* agent (in yellow) has been created. This agent is close enough to the existing *building block* agent (in green) to merge with it. The resulting agent will be composed of the 5 *building* agents composing the two *building block* agents.

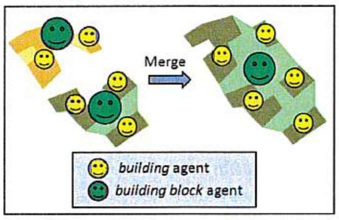

Fig. 6. Fusion of different emergent agents

The purpose of the "disposal" operation is to express when an emerging structure should not consider to be an agent in the simulation anymore. The emergent agent representing the structure is cleared out of the simulation and its components become free.

Figure 8 illustrates the "disposal" operation. Following the example presented Figure 7, we consider that three of the *building* agents composing the *building block* agent died. Now, the remaining *building* agents are too far from each other to compose a *building block* agent. Then, the *building block* agent is going to die.

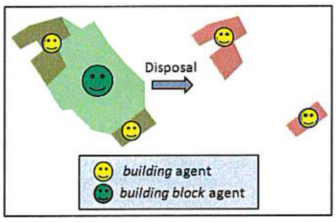

Fig. 7. Death of an emergent agent

The top-down feedback control allows the modeler to describe which feedback constraint an emergent agent is exercising on its underlying micro-agents. As emergent agents usually emerge because of the interactions of certain micro-agents, these agents have an influence on its attributes and behavior. Reciprocally, an emergent agent may also provide a feedback on the behavior of its components, either implicitly or explicitly. In order to describe it, the modeler needs to have some way to alter the behavior of a micro-agent (by changing parameters, adding, or removing entire behaviors) before and after it enters an emergent agent.

Typically, in our city dynamic simulation example, a *building* agent, once part of *building block* agent, has more chance to attract residents to live in, and thus to lead to construction of new buildings in the neighborhood (for example, shops).

3.2.2 Representing emergent agents in GAMA

An emergent agent is composed of constituent agents. Constituent agents can be considered as micro-agents compared to the emergent agent. Reciprocally, the emergent agent can be seen as a macro-agent compared to its constituent agents. In turn, several emergent agents can be merged to form another emergent agent at a higher level of abstraction. Thus, an agent in GAMA can play the role of macro-agent in one level of organization and micro-agent in a higher level of abstraction. This design aims at permitting the modeler to represent as many levels of abstraction as he needs in his model. Figure 9 shows an example of abstraction level hierarchy for the city dynamic simulation problem: a *city* agent is composed of a set of *district* agents that are each composed of a set of *building block* agents that are at their turn composed of a set of *building* agents.

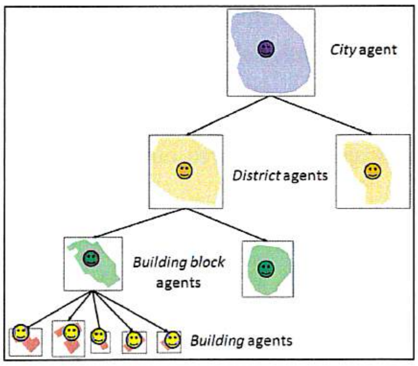

Fig. 8. Example of levels of abstraction hierarchy

To manipulate the five specific operations in the lifecycle of an emergent agent (create, update, merge, disposal, top-down constraint control), six GAML commands are defined: *creation*, *update*, *merge*, *disposal, enable* and *disable.*

- The *creation* command allows to specify when emergent agents are created in the simulation.

  Example: the following GAML lines create a *building block* agent which has for components the *building* agent contained in the list *list_buildings*:

```
<creation>
   <create with="[components::list_buildings]"
    species="building" />
</creation>
```

- The *update* command allows the modeler to define how the constituent micro-agents are added and removed from an emergent agent.

  Example: the following GAML lines update the components of the *building block* agent that is applying this command by adding the *building* agents contained in *added_buildings* and removing the ones contained in *removed_buildings*:

```
<update>
  <set name="components" value ="components + added_buildings -
      removed_buildings"/>
</update>
```

- The *merge* command allows the modeler to define how several emergent agents are merged.

  Example: the following GAML lines allow to merge several *building block* agents (the ones contained in the *nearby_bb* list) with the *building block* agent applying this command. All the constituent *building* agents of the *building block* agents contained in the *nearby_bb* list are added to the component list of the one applying the command. Then, the other *building block* agents die (i.e. are removed from the simulation):

```
<merge>
  <loop over="nearby_bb" var="one_bb">
     <set name="components" value ="components +
       one_bb.components"/>
     <ask target="one_bb">
        <do action="die">
     </ask>
  </loop>
</merge>
```

- The *disposal* command allows the modeler to specify when an emergent agent is cleared out of the simulation.

  Example: the following GAML line specifies that a building *block agent* will be removed from the simulation if it contains less than two *building* agents:

```
<disposal when="(length components) < 2"/>
```

- The *disable* command allows the modeler to disable certain behavior units appropriately. While the *enable* command allows the modeler to enable the inactive behavior units.

  Example: the following GAML lines enable the behavior "expansion" and disable the behavior "destruction" of the *building* agent *one_building_agent*:

```
<ask target="one_building_agent">
   <enable behavior="'expansion'">
   <enable behavior="'destruction'">
</ask>
```

Note that GAMA provides several clustering algorithms (e.g. hierarchical clustering, X-Means [23], etc.) that can be used to dynamically detect if an emergent agent has to be instantiate. For example, these algorithms can be used to detect groups of close agents, or agents sharing some specific attributes.

Example: the following GAML lines allows to regroup the *building* agents contained in the *buildings* list into a set of groups; each group being composed of *building* agents of which the distance to each other is lower or equal to 10m:

```
<do action="simple_clustering_by_distance"
    return="groups">
   <arg name="agents" value="buildings" >
   <arg name="dist_max" value="10m" >
</do>
```

4. Use of GAMA for real projects

The last version of GAMA is already used in many projects concerning different domains of applications: avian flu local propagation in North Vietnam [3], the rift valley fever in Senegal, the brown hopper invasion in South Vietnam [4] the emergency response in Hanoi [5], etc. In this section, we propose to present in details two of these projects.

4.1 GAMAVI

H5N1 is still a major threat for both economy and health, in particular in South East Asia and its eradication is far from being achieved. The goal of the GAMAVI model is to study the persistence of the H5N1 in the environment and the relationships between environments (as virus reservoirs) and the traditional or semi-commercial poultry production systems. The model focuses on farms and poultry flocks in a village and several environments are represented: building, inner-village ground, road, rice-field (flooded or dry) and pond. Figure 10 shows a snapshot of the model implemented with the GAMA platform.

In the model, the village itself and its rice-field and surroundings are agents created from shapefiles. Flock agents can wander inside the village or move to an objective (rice-fields, ponds and farms). These movements used the capacities of GAMA concerning the computation of a shortest path inside a polygon. The collecting and depletion of the virus in the environment is done thanks to a fine-grained grid called the viro-grid. Basic geometrical comparisons are conducted between this grid and the GIS data to define the natural environment type of the cells. Concerning the collection of the virus, the excreting flocks update an *excretion* variable of the cell; at the end of a time-step a behavior updates the current concentration according to this variable; finally a daily behavior is executed in order to compute the depletion of the virus level. More details about the model can be found in [3]. In the current version of the model, only one level of agents is considered: the flock. In its next version, each individual poultry will be considered as an agent and the flock will be considered as a macro-agent. This new modeling of the agents will allow to give more complex behaviors to the agents and thus to improve the realism of the model.

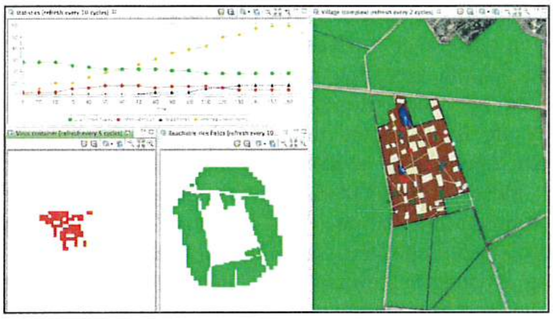

Fig. 9. GAMAVI model

4.2 AROUND/ISSUE

The problem of emergency responses to disasters is a very serious and complex social issue. It involves a large number of heterogeneous actors that have to work together in a hostile environment. In particular the decision-making process of each stakeholder and the coordination between them are quite hard to model precisely.

The aim of the ISSUE model is to simulate the relief effort and to learn human strategies from various disaster scenarios. The devastated infrastructures and human casualties are input GIS data for the rescue simulation. Rescue teams, such as ambulances, fire-fighters or policemen are modeled and as agents. These agents are moving along the road network and can communicate with each other to define a rescue strategy. In the same way, the roads and the buildings are modeled as agents. This agentification allows to give a dynamic behavior to the roads and buildings during the simulation: e.g. a building impacted by an earthquake can collapse during the simulation and block a road. Figure 5 shows a snapshot of the model implemented with the GAMA platform. More details about the model can be found in [5]. In the next version of the model, the multi-level modeling capabilities of GAMA will be used to create dynamic groups of rescuers able to organize themselves to improve their efficiency.

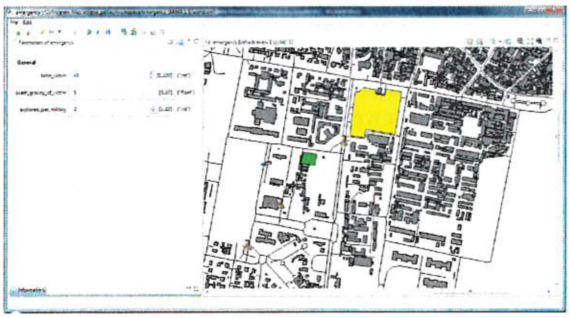

Fig. 10. ISSUE model

5 Discussion

We see the contributions of this work as threefold:

1- There is a difference between an idea and its implementation. What we incorporate into GAMA are implementations of ideas that may have been (or not) already proposed by other people but rarely found their way into operational instances. They are implemented into the platform and linked with the modeling language, so that they can be used by anyone building a model in GAMA. In our point of view, these implementations are contributions to the field, because they eliminate the ambiguities and the lack of formalism often found in ABM/MAS contributions and, most important, can be experimented.

2- Integrating existing techniques in a framework and enabling the researchers to easily choose the most appropriate is a delicate exercise. In GAMA, we have ensured that all the proposed techniques are tightly coupled, and that they are usable even by novice users through GAML. This allows us to build, in the same platform, simple models (a la NetLogo) alongside more complex models. Actually, our efforts of integration tend to the point that there is no real difference between a "simple" and a "complex" model. So, while it is true that, for instance, we did not invent graph-related techniques, we believe we contribute to the field by providing a way, for researchers, to use the most appropriate techniques, transparently, into their models.

3- Following the previous point, we see GAMA as a contribution by itself, filling the gap between NetLogo, interesting for prototyping small models, but which does not scale well when it comes to real ones, and RePast, more a complete toolbox than a platform. The fact, for instance, that every agent in GAMA is provided with a geometry, and that any environment can be discretized, means that researchers can begin with a simple prototype (where agents are points on a grid, like in Netlogo) to test the logic of a model, and turn this model into a more realistic one, for example by loading data from a GIS base, without having to change anything to the logic. This radically transforms the experimental processes of ABM.

6 Conclusion

In this paper, we present the new advance features included in the last version of the GAMA platform (version 1.3) [2]. These features concern the use of geographic vector data and the definition of multi-level models.

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, the rift valley fever in Senegal, the brown hopper invasion in South Vietnam, the effect of emotions on waves of panic.

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 (November 2010).
3. Amouroux, E., Taillandier, P. and Drogoul, A., Complex environment representation in epidemiology ABM: application on H5N1 propagation. *ICTACS*, (2010).
4. 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).
5. 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)
6. Taillandier, P., Buard, E.: Designing Agent Behaviour in Agent-Based Simulation through participatory method. *PRIMA2010, Nagoya, Japan*, 571-578 (2009)
7. Crooks, A.T.: Constructing and implementing an agent-based model of residential segregation through vector GIS. *IJGIS*. Vol. 24, No. 5, 661–675 (2010)
8. 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)
9. 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)
10. Box, P.: Spatial Units as Agents. *Integrating GIS and Agent-Based Modelling Techniques*. ed. Oxford (2002)
11. 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)
12. Wilensky, U. NetLogo. http://ccl.northwestern.edu/netlogo/. *Center for Connected Learning and Computer-Based Modeling*, Northwestern University. Evanston, IL (1999)
13. Russell, E., Wilensky, U.: Consuming spatial data in NetLogo using the GIS Extension. *The annual meeting of the Swarm Development Group*. Chicago, IL (2008)
14. Bousquet, F., Bakam, I., Proton, H. Le Page, C.: Cormas: common-pool resources and multi-agents systems. *IEA/AIE* (Vol. 2) 826-837 (1998)
15. Urbani, D., Delhom, M.: Analyzing Knowledge Exchanges in Hybrid MAS GIS Decision Support Systems, Toward a New DSS Architecture, *LNCS 4953*, 323-332 (2008)
16. 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)
17. 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)
18. Dijkstra, E. W.: A short introduction to the art of programming. *Technological Univ. Eindhoven*, Rep. EWD316 (1971)
19. Floyd, R.W.: Algorithm 97: Shortest Path, *Communications of the ACM*, vol. 5, n° 6, p. 345 (1962)
20. Camazine, S. et al.: Self-Organization in Biological Systems. *Princeton University Press*, Princeton (2001).
21. Schelling, T.: Schelling's Segregation model, http://web.mit.edu/www/lab/alife/schelling.html (September 2010)
22. Laurent Breton et al.: A Multi-Agent Based Simulation of Sand Pile in Static Equilibrium. *MABS2000* (2000)
23. Pelleg, D., Moore, A. W.: X-means: Extending K-means with Efficient Estimation of the Number of Clusters. *ICML*, 727-734, (2000)