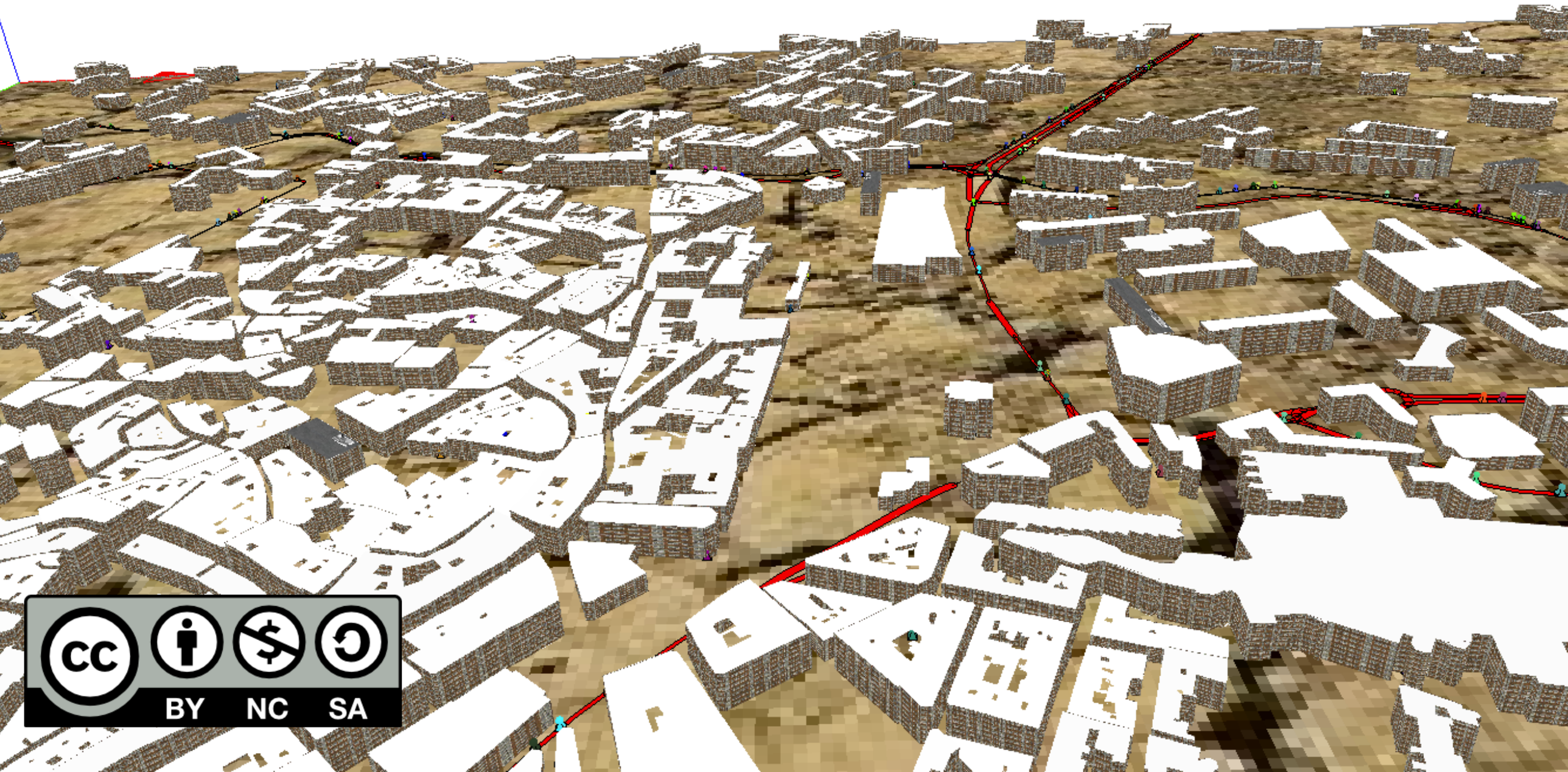
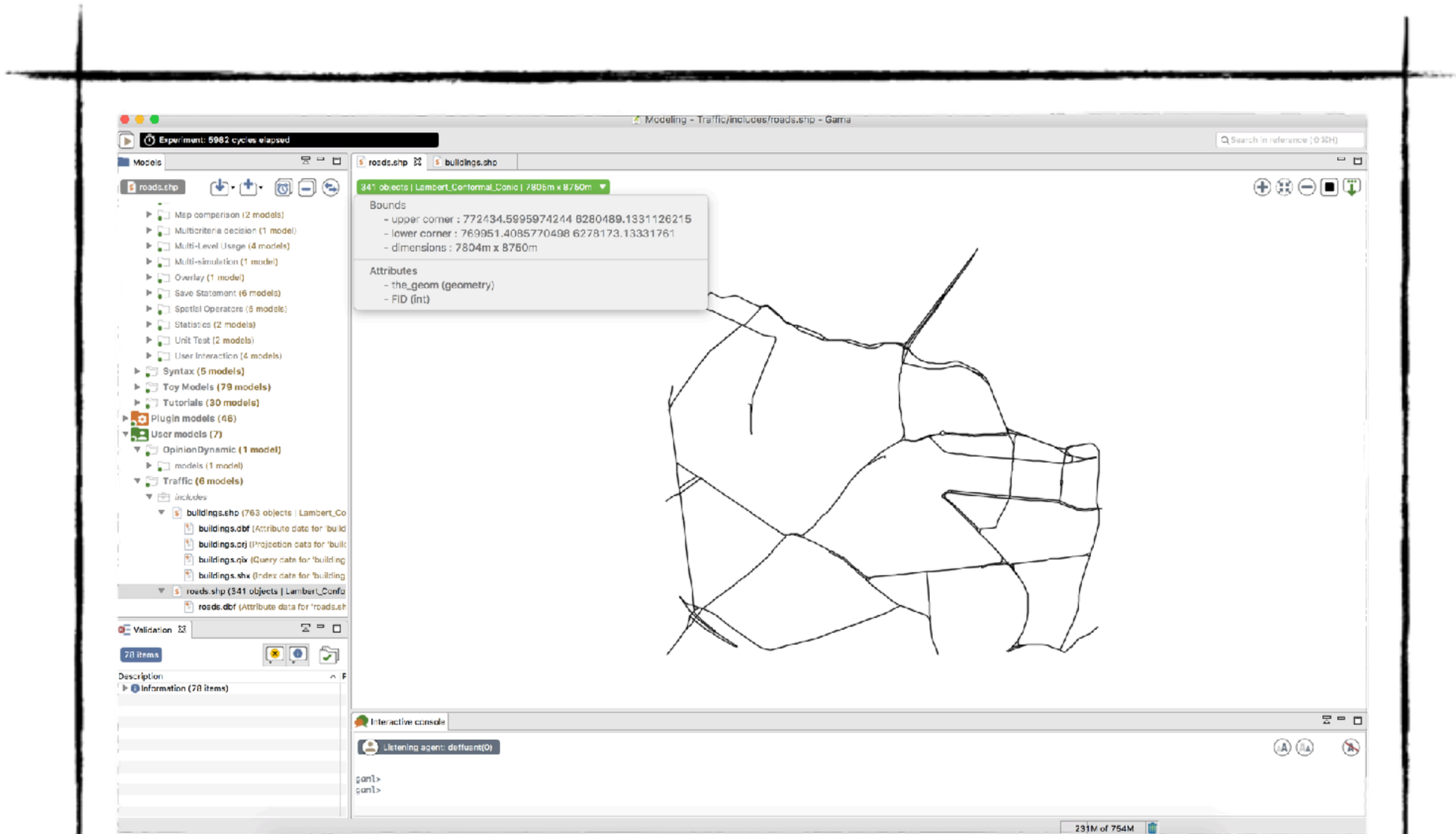
Introduction to GAMA-platform
Toward a traffic model

Benoit GAUDOU, *IRD UMMISCO, University Toulouse 1 Capitole, USTH*
Arthur BRUGIERE, *IRD UMMISCO*
Doryan KACED, *IRIT, University Toulouse 1 Capitole*
NGUYEN Ngoc Doanh, *Thuy Loi University, WARM team, UMI UMMISCO*



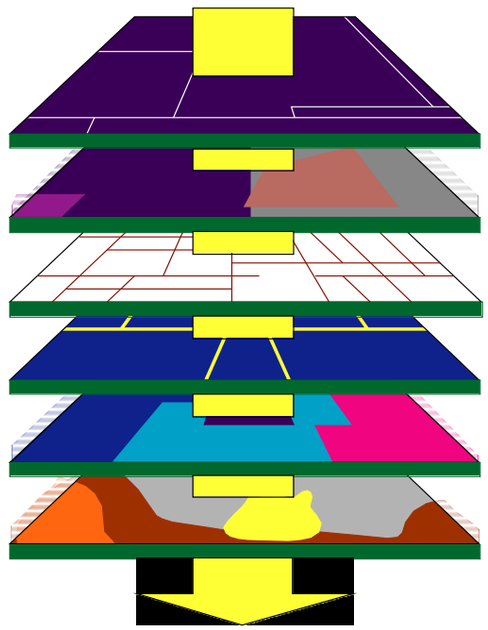
Introduction to GIS data and its management in GAMA

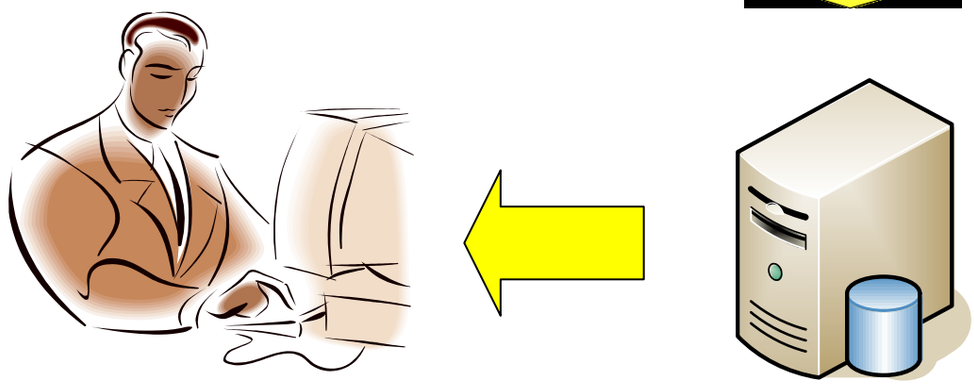


What are Geographic Information Systems (GIS)?

- A computer system designed to capture, store, manipulate, analyse, manage, and present all types of geographical data.

The real world

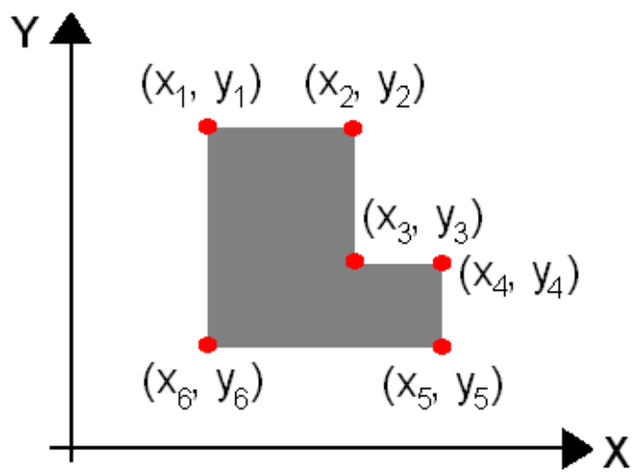




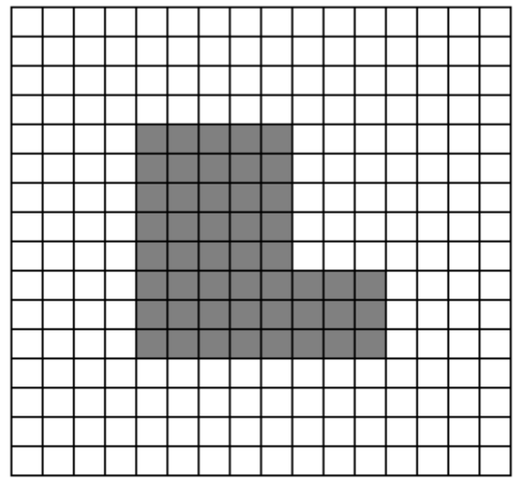
Database

Representation of geographical data

- **Raster** (grid or image): A geographical phenomena is represented as a partition (in cells) of the geographical space. Each cell has one or several attributes that define its content.



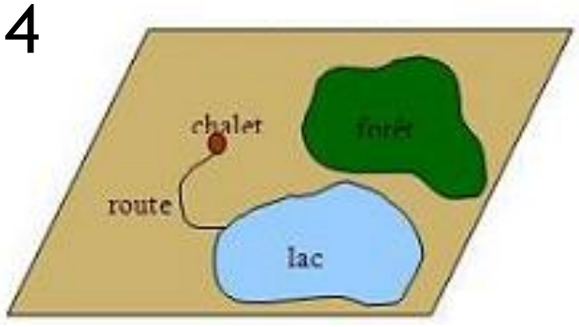
Vector



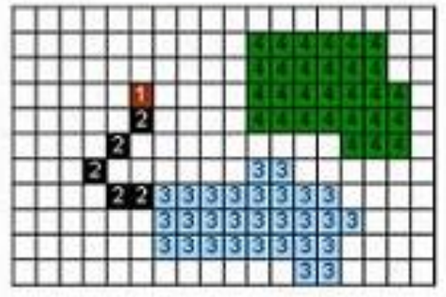
Raster

- **Vector**: A geographical phenomena is represented by one or several geometric primitives (point, line, polygons), described by a list of coordinates and an interpolation function. A classic vector format filed is the shapefile: it is composed of 4 main files: .shp, .dbf, .shx, .prj

Real world



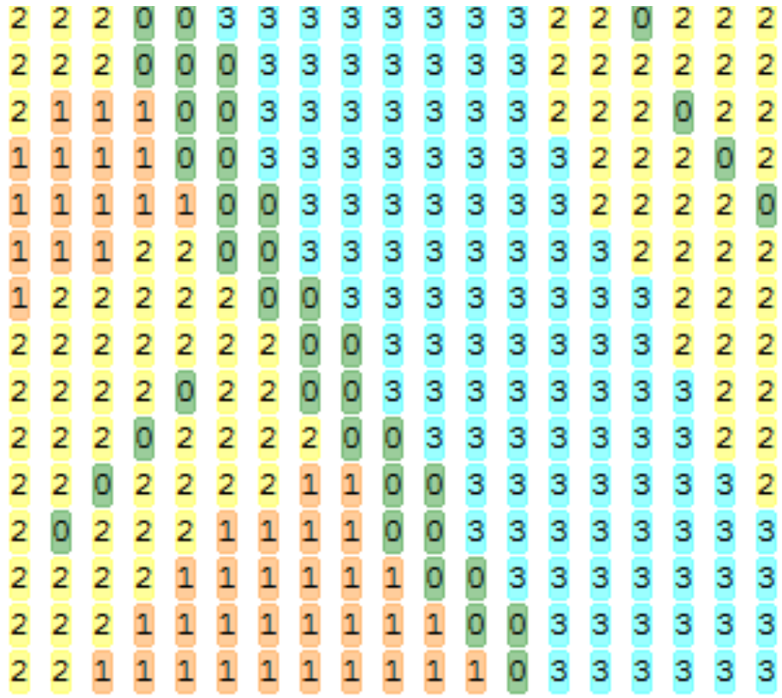
Vector



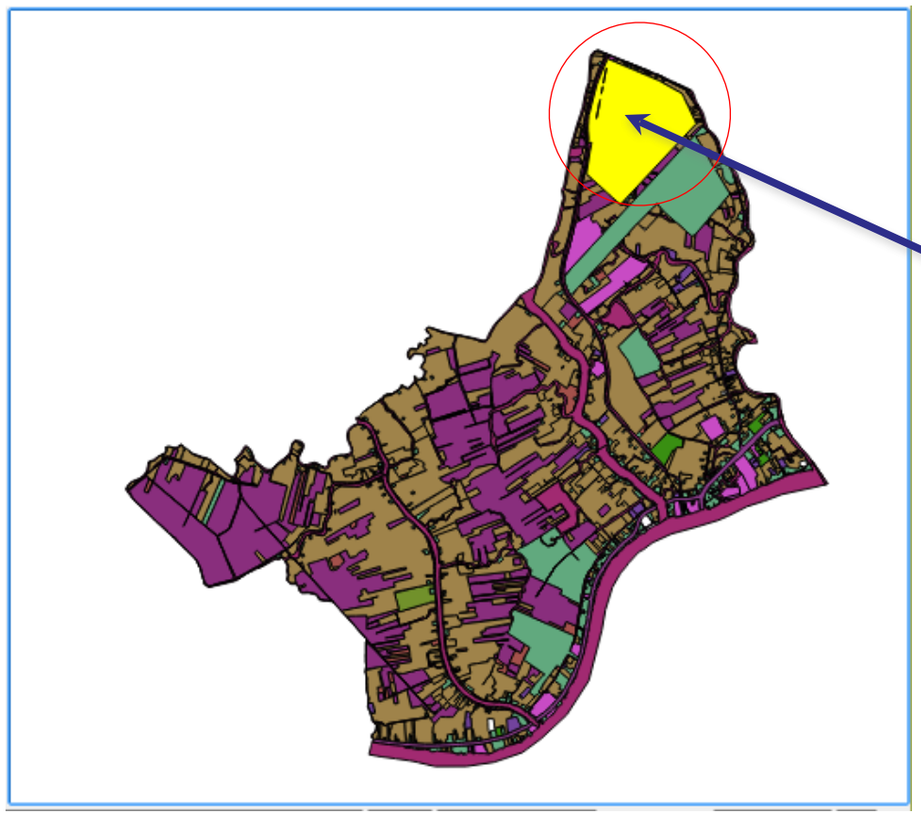
Raster

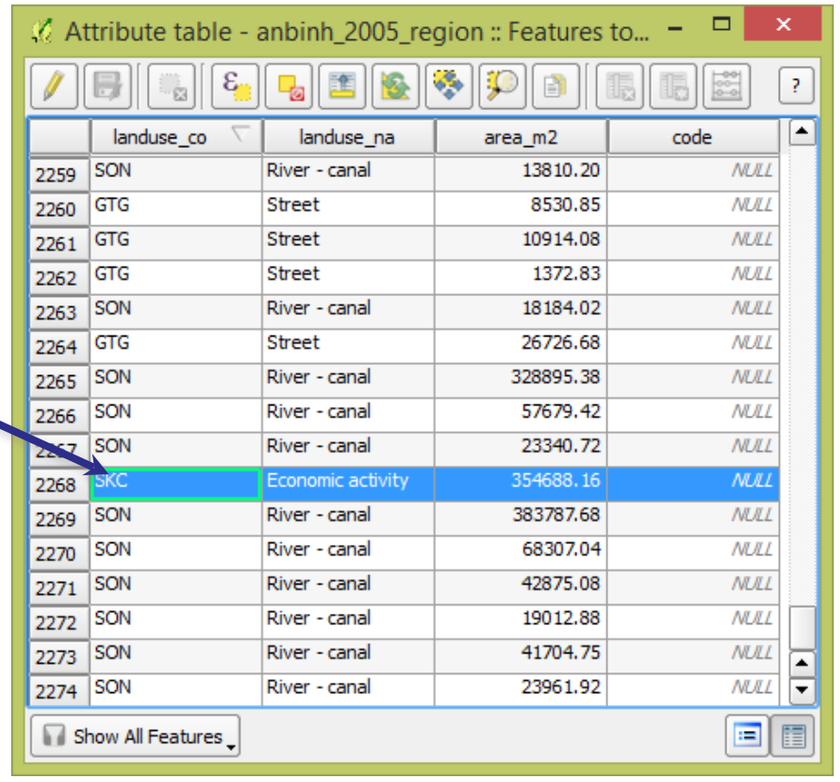
Attribute data - non spatial data

- In raster datasets, information associated with each unique value of a raster cell.
- In vector dataset, data stored in a table and linked to each object by a unique identifier



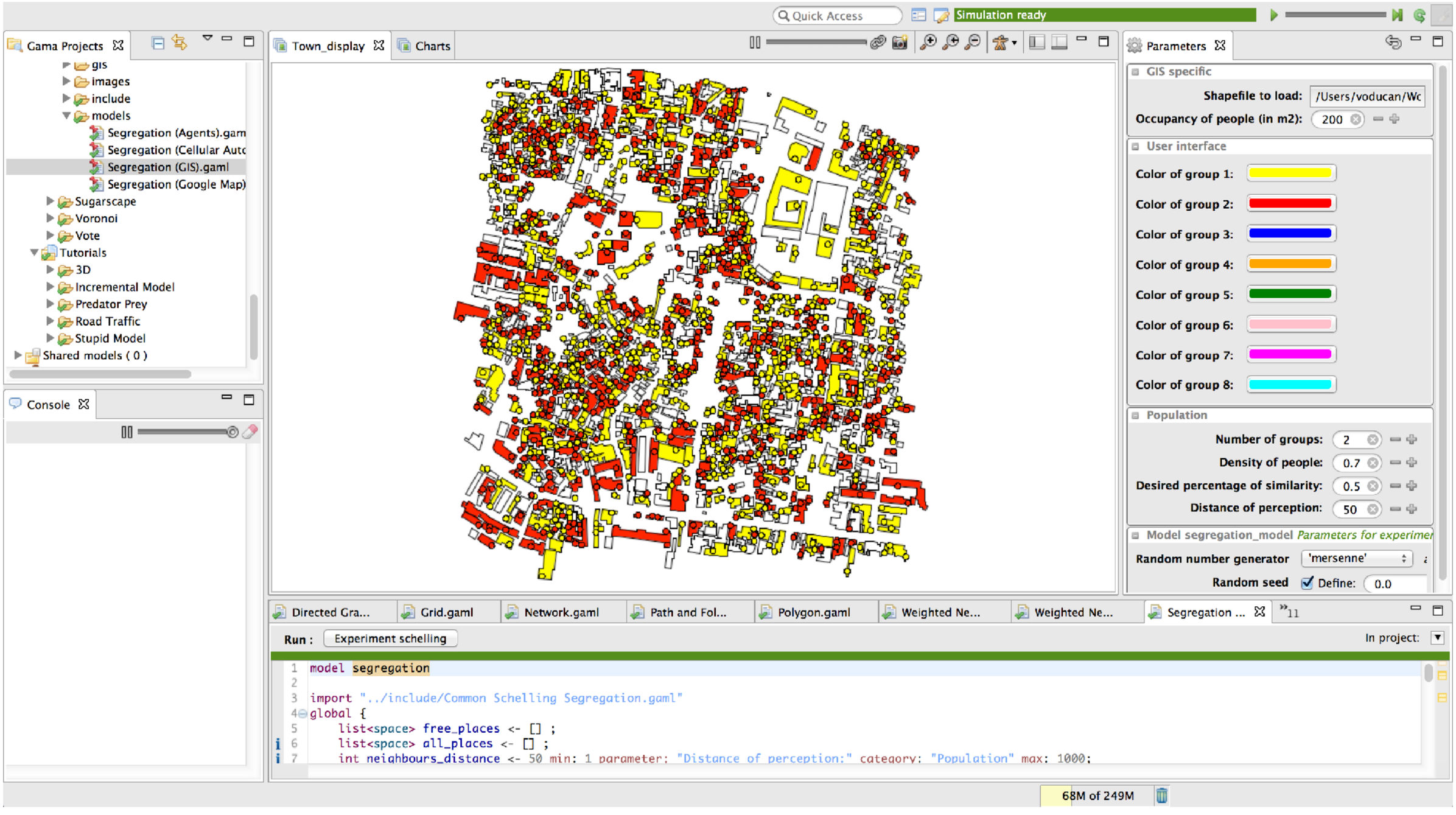
Attribute of a raster format



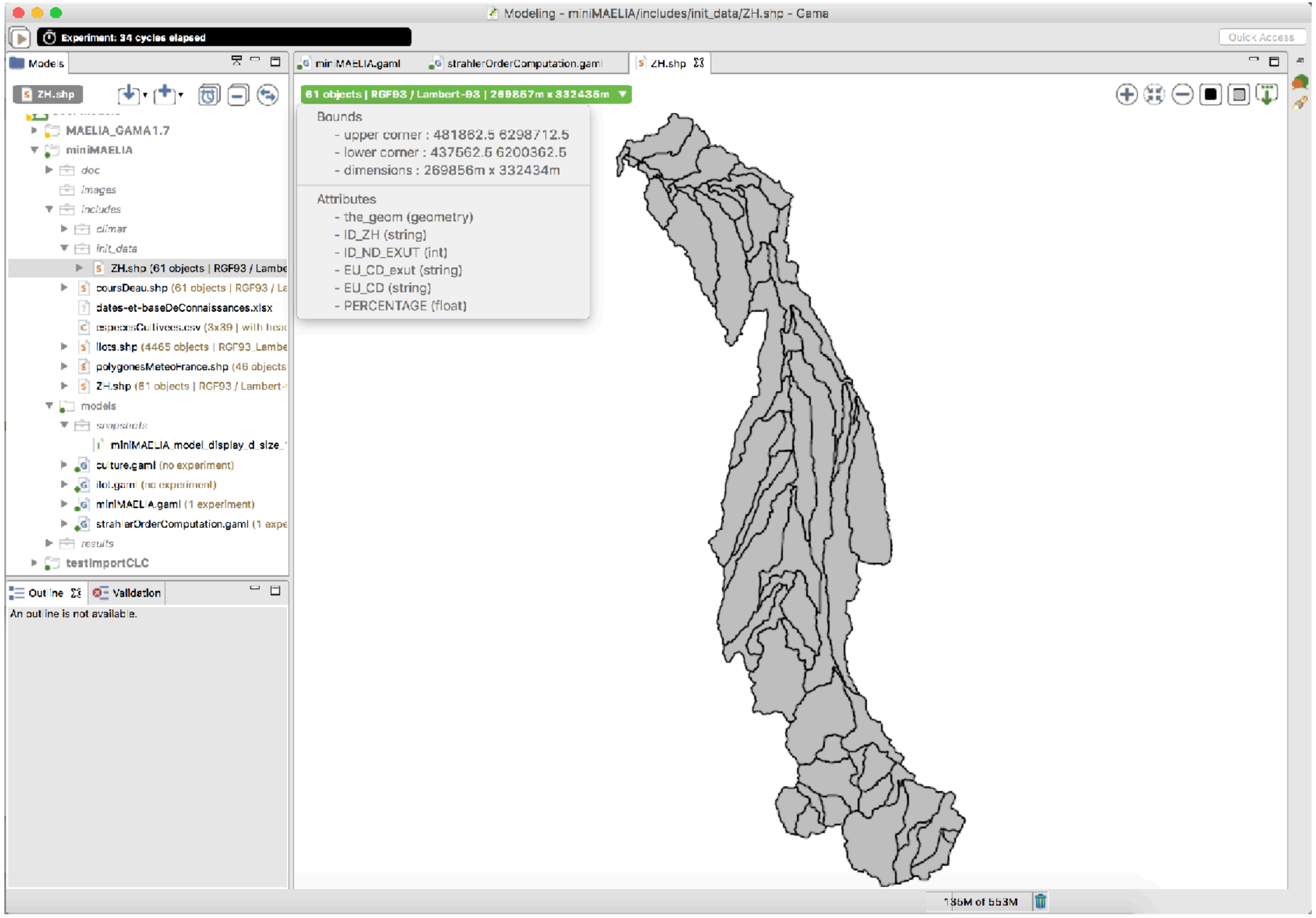


Attribute of vector format

GAMA provides many features to manage GIS data and vector geometries



GAMA allows modellers to display shape files and attributes information

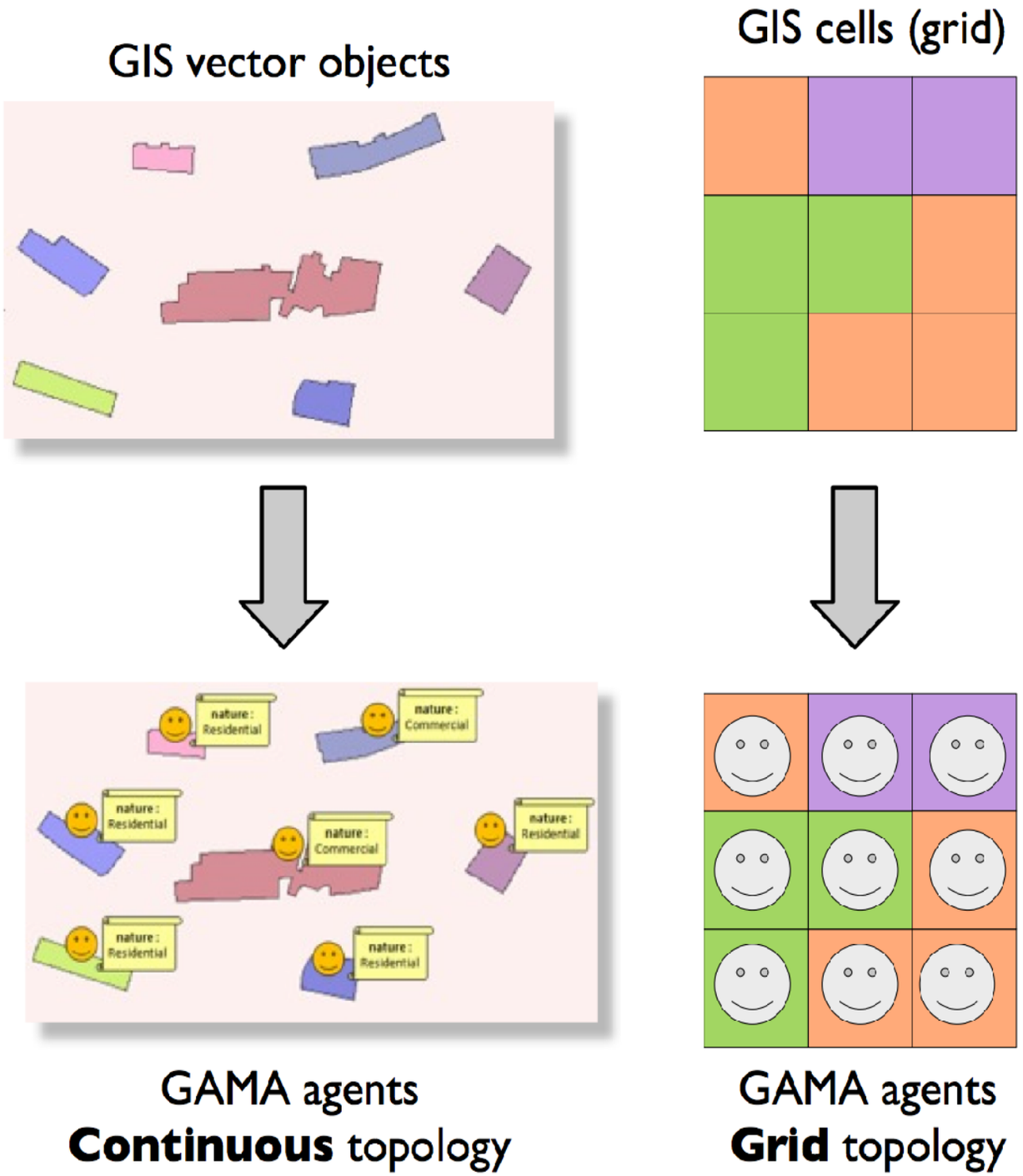


Every agent in GAMA has a geometry (its shape).

An agent's geometry can be:
- a **point** (default),
- a **polyline**,
- a **polygon** or
- a complex geometry (2D-3D)

An agent geometry is accessible through GAML thanks to the "shape" built-in attribute.

World agent (<<global>> species) have also a "shape" built-in attribute which define the shape of environment



GAMA agents
Continuous topology

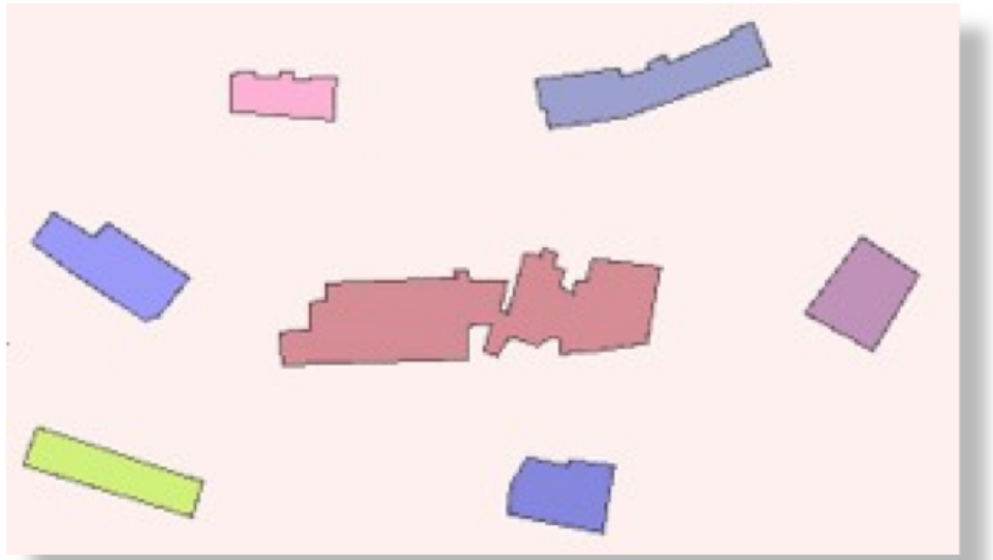
GAMA agents
Grid topology

Agents can be created directly from GIS shapefile

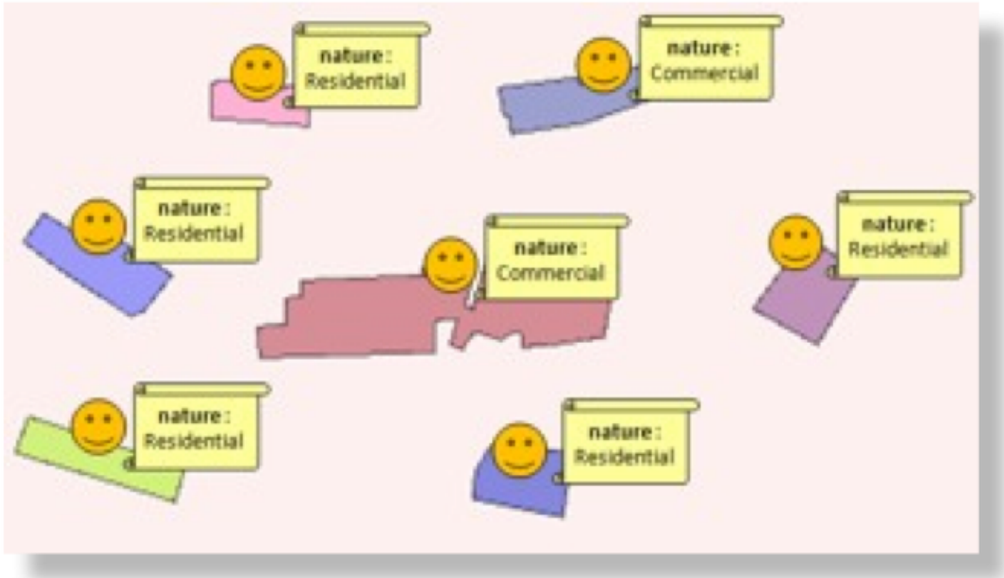
create my_species from: my_shapefile;

Each GIS object becomes an agent of the specified species

my_shapefile

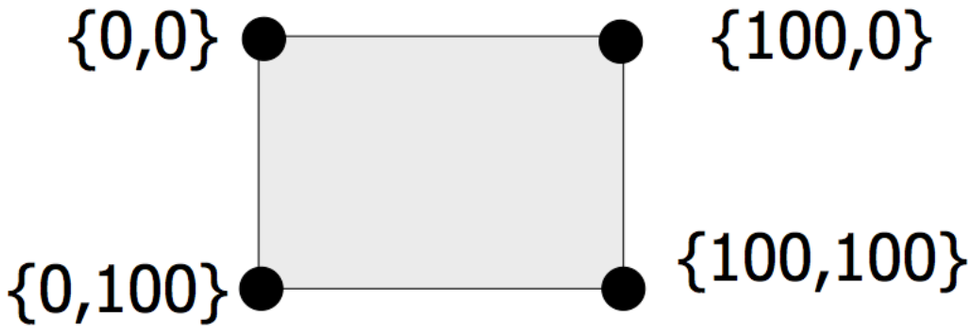


my_species



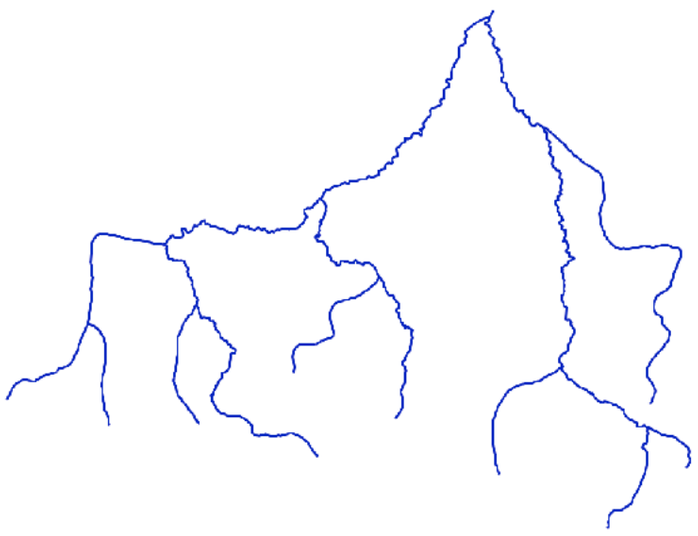
The shape of the “world” agent represents the global environment of a model

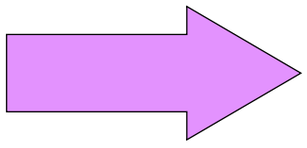
By default, a model global environment is a 2D square space of 100m x 100m

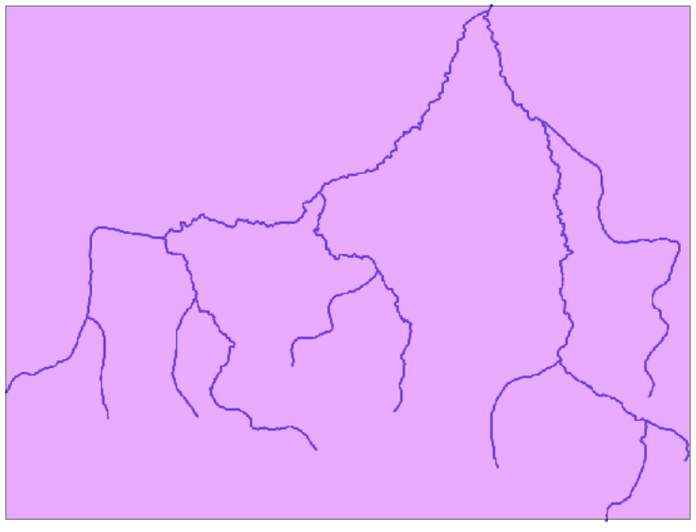


It is also possible to redefine the global environment using a shapefile or asc file.

```
global{
    file river_shapefile <- file("../includes/river.shp");
    geometry shape <- envelope(river_shapefile);
}
```

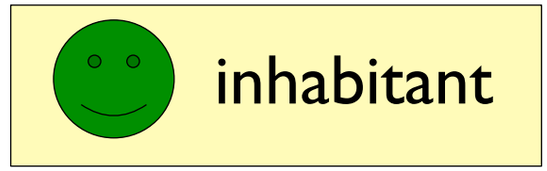






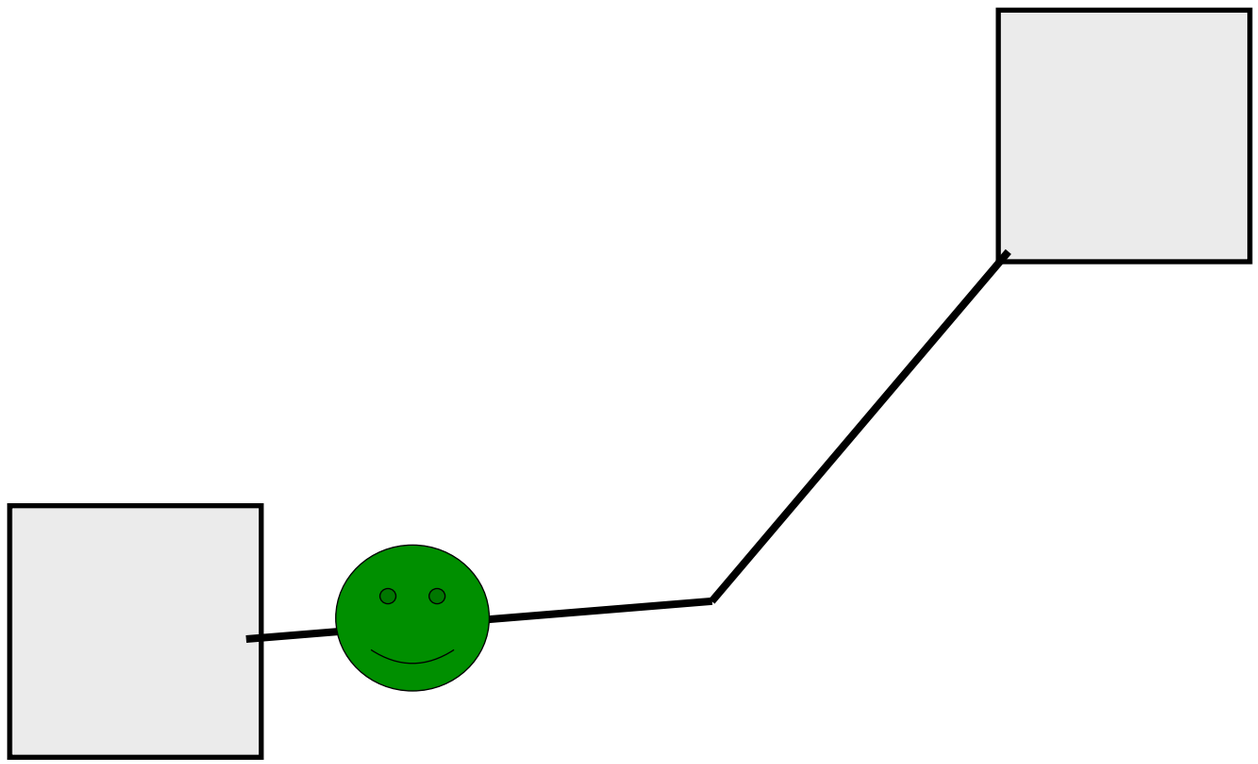
Environment

Introduction to a simple traffic simulation



Inhabitants move from one building to another one on a road network.

In a building, they will stay for some time: at each simulation step, they have a probability to leave (`proba_leave`)



Step 1: definition of building and road agents

Objectives:

- Definition of building and road species
- Creation of building and road agents
- Display agents

Definition of the *building* species

TO DO: define the *building* species, with an aspect called “geom” drawing the shape of the agent with a grey color.

```
model my_model

global {
}

species my_species {
}

experiment my_model type: gui {
}
```

Answer:

```
species building {
   aspect geom {
      draw shape color: #gray;
   }
}
```

The agent geometry is accessible get through the ***shape*** attribute.

Definition of the *road* species

TO DO: define the *road* species, with an aspect called “geom” drawing the shape of the agent with a black color.

Answer:

```
model my_model

global {
}

species my_species {
}

experiment my_model type: gui {
}
```

```
species road {
   aspect geom {
      draw shape color: #black;
   }
}
```

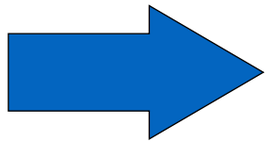
We have defined the building and road species.
Next step: creation of the *building* and *road* agents !

Definition of the shape files

TODO: define 2 new global variables (with the file type) getting as value the shape file of buildings (resp. roads). Use the last on to redefine the environment size.

Answer:

```
global {
    file shapefile_buildings <- file("../includes/buildings.shp");
    file shapefile_roads <- file("../includes/roads.shp");
    geometry shape <- envelope(shapefile_roads);
    ...
}
```



Environnement

```
model my_model

global {
}

species my_species {
}

experiment my_model type: gui {
}
```

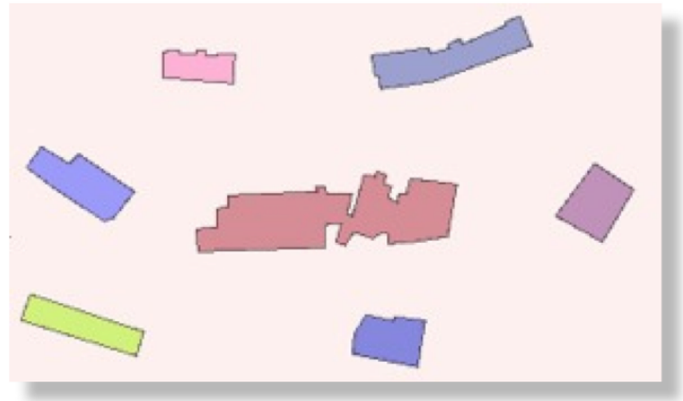
Creation of building and road agents

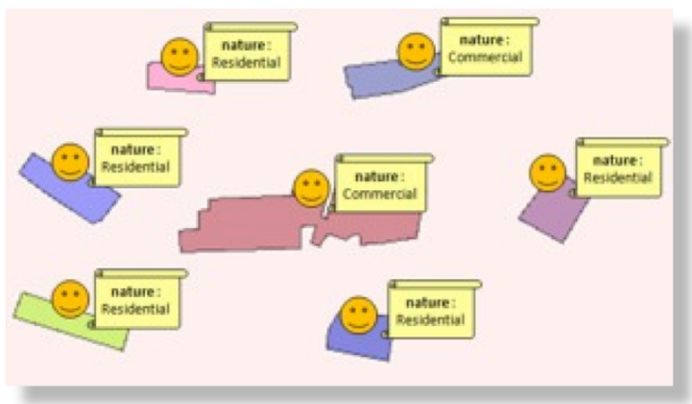
TODO: define an *init* section (in the global) to create building and road agents from the 2 shapefiles.

```
model my_model

global {
}

species my_species {
}

experiment my_model type: gui {
}
```

Answer:

```
global {
    //variables
    init {
        create building from: shapefile_buildings;
        create road from: shapefile_roads;
    }
    ...
}
```





We have created building and road agents.
Next step: display them !

Display building and road agents

TODO: add *building* and *road* agents in a display (named “map”) with their aspect.

```
model my_model

global {
}

species my_species {
}

experiment my_model type: gui {
}
```

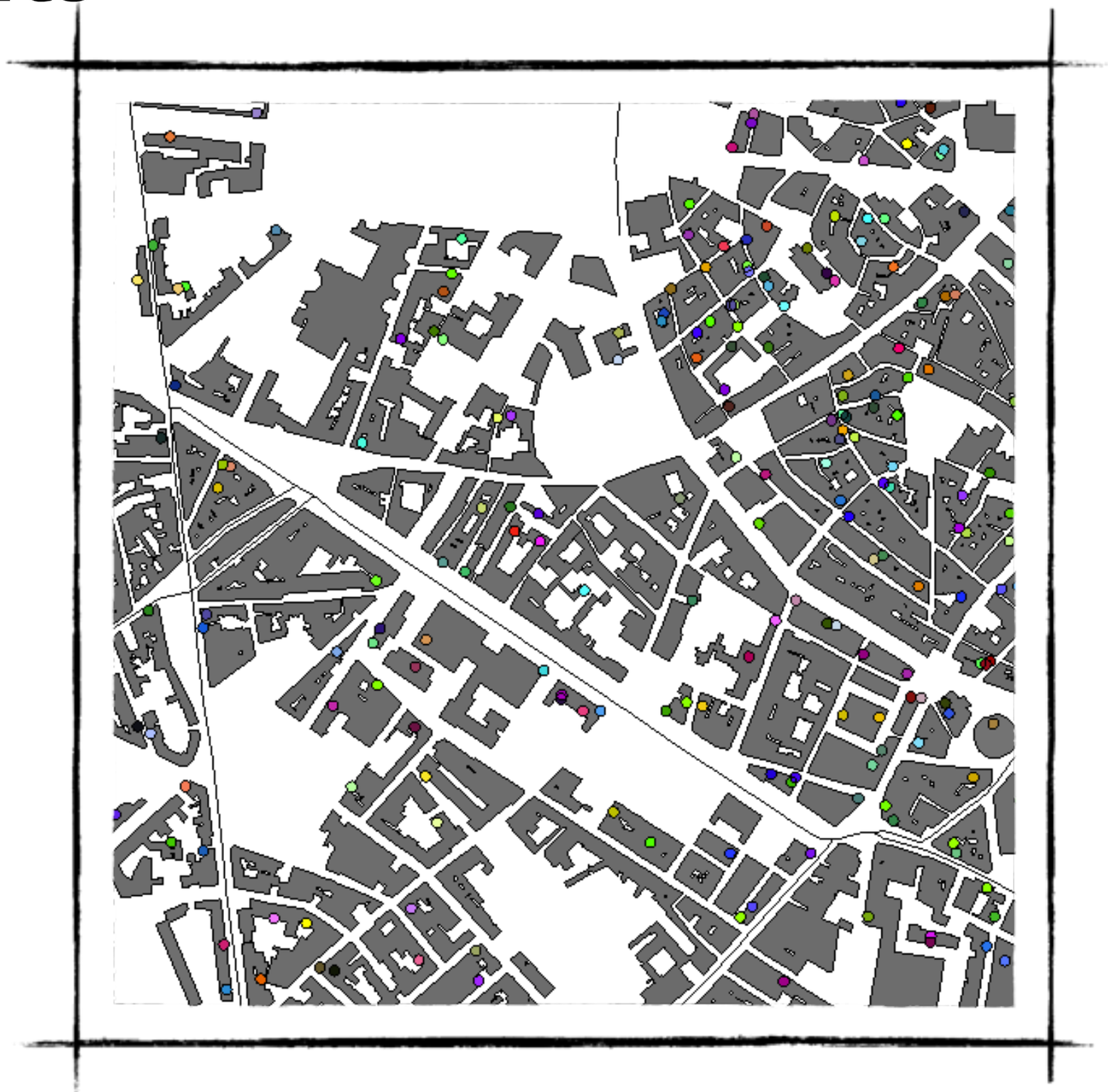
Answer:

```
experiment traffic type: gui {
    output {
        display map {
            species building aspect: geom refresh: false;
            species road aspect: geom;
        }
    }
}
```

End of step 1

We will define now the *inhabitant* species.

Step 2: definition of inhabitant agents



Objectives:

- Definition of the *inhabitant* species
- Creation of *inhabitant* agents
- Display agents

Definition of the inhabitant species

TODO: define an *inhabitant* species with the moving skill and 3 attributes:

- *target* (type = point)
- *proba_leave* (type = float, init value= 0.05)
- *speed* (type = float, init value = 5 km/h)
- *color* (type = rgb, init value = random color)

Its aspect will be a circle (radius = 5m) with the color *color*.

Answer:

```
species inhabitant skills: [moving]{
    point target;
    rgb color <- rnd_color(255);
    float proba_leave <- 0.05;
    float speed <- 5 #km/#h;

    aspect circle {
        draw circle(5) color: color;
    }
}
```

```
model my_model

global {
}

species my_species {
}

experiment my_model type: gui {
}
```

A *skilll* is a plugin (written in Java) giving new variables and actions to agents.

With the *moving* skill, agents get new attributes (*speed*, *heading*, *destination*) and actions (*follow*, *goto*, *move*, *wander*) supplémentaires

the operator **rnd_color**(255) returns a random color

Creation of inhabitant agents

TODO: create 1000 inhabitant agents and locate them in a building.

Answer:

```
model my_model

global {
}

species my_species {
}

experiment my_model type: gui {
}
```

```
global {
    //variables
    init {
        //creation of buildings and roads

        create inhabitant number: 1000{
            location <- any_location_in(one_of(building));
        }
    }
    ...
}
```

the operator **any_location_in(a_geometry)** returns a random location inside a geometry

Display of inhabitant agents

TODO: add inhabitants to the map display with their circle display.

Answer:

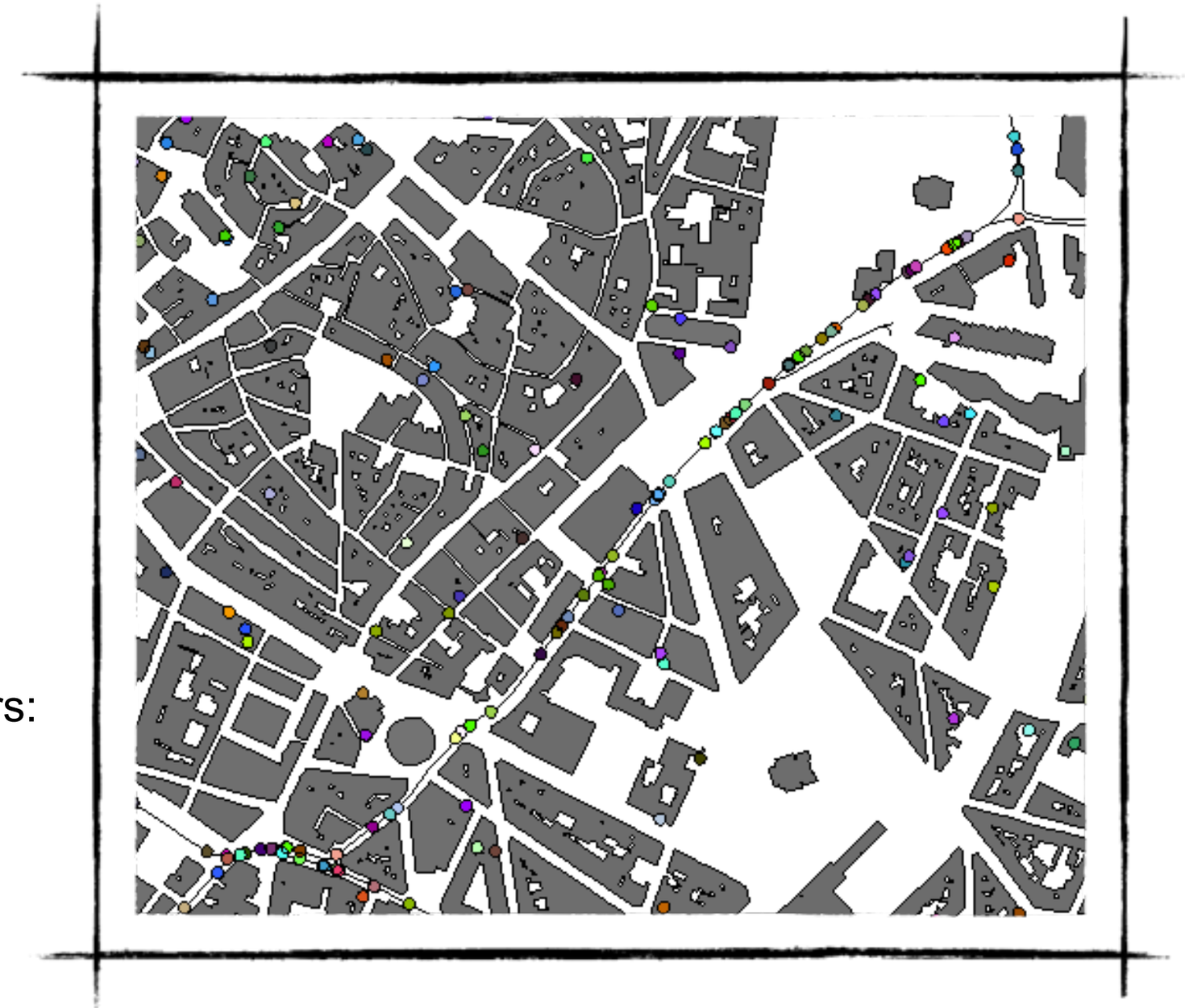
```
experiment traffic type: gui {
    output {
        display map type: opengl{
            species building aspect: geom refresh: false;
            species road aspect: geom refresh: false;
            species inhabitant aspect: circle;
        }
    }
}
```

```
model my_model

global {
}

species my_species {
}

experiment my_model type: gui {
}
```

End of step 2

Now we will define inhabitant agents' behaviour.

Step 3: definition of inhabitant agents' behaviour



Objectives:

- Creation of the road network
- Creation of *inhabitant* agents' behaviours:
  - leave buildings
  - move on the graph

Definition of the road network

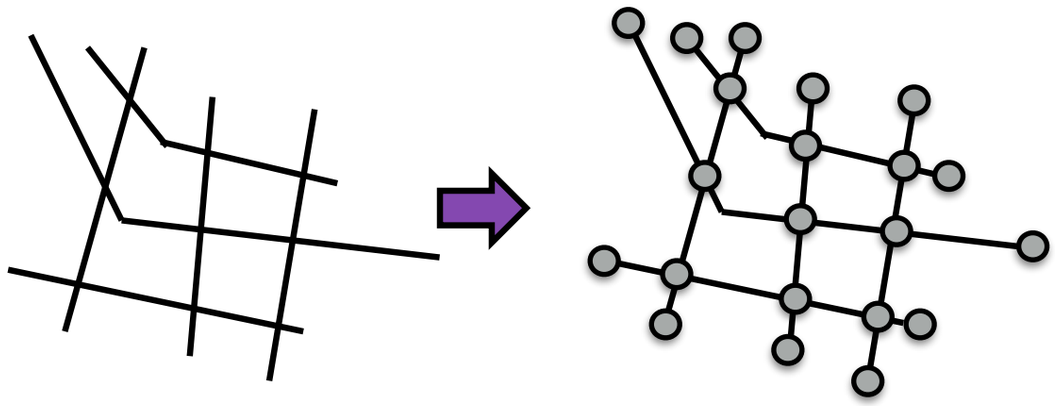
```
model my_model

global {
}

species my_species {
}

experiment my_model type: gui {
}
```

TODO: define a new global variable road_network and initialise it in the global init block (with a graph created from the road agents).

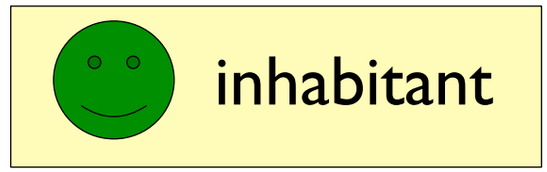
Answer:

```
global {
    // other variables
    graph road_network;
    init {
        create building from: shapefile_batiments;
        create road from: shapefile_routes;
        create inhabitant number: 1000{
            location <- any_location_in(one_of(building));
        }
        road_network <- as_edge_graph(road);
    }
}
```

The operator **as_edge_graph(***list of polylines***)** builds a graph from a list of polylines.



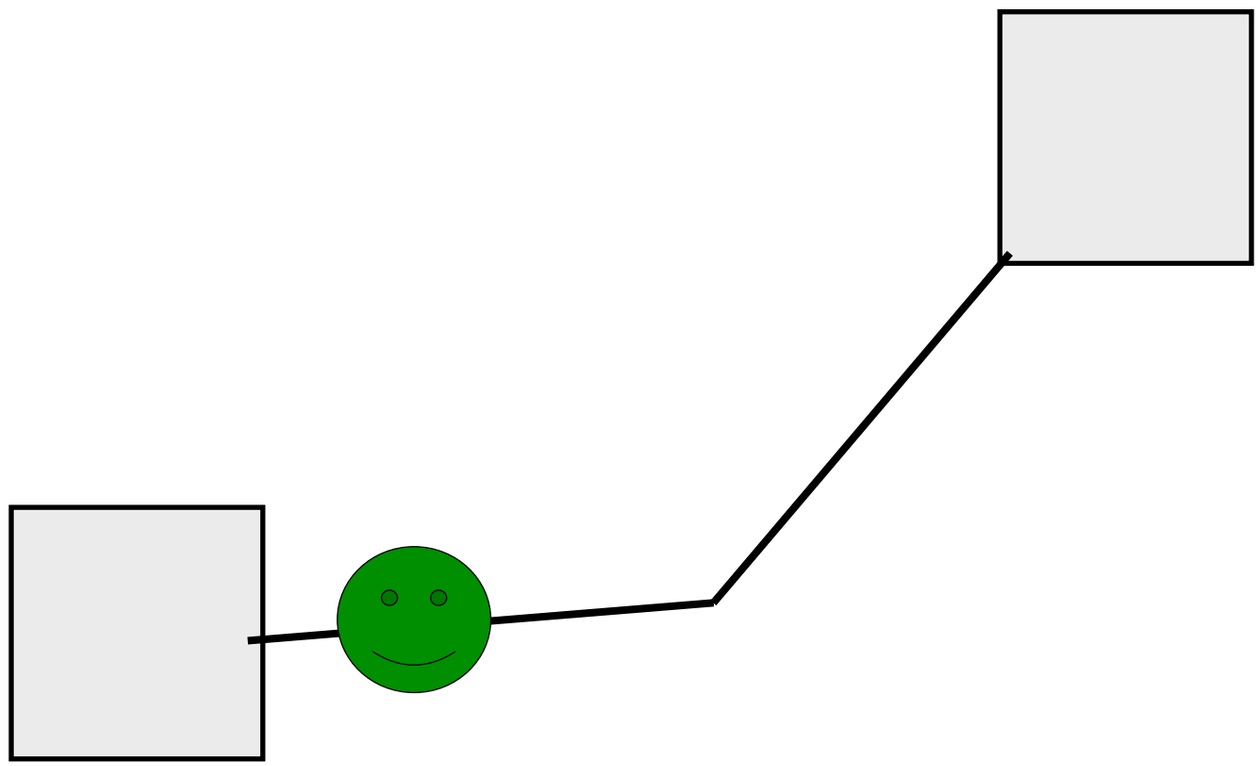
Inhabitant agents' behaviour



Inhabitants move from one building to another one on a road network.

They move if and only if they have a target.

In a building, they will stay for some time: at each simulation step, they have a probability to leave (`proba_leave`)



inhabitant species: *leave* reflex

TODO: define a new reflex (named leave) for inhabitant species:

- it is activated whether the agent does not have a target and given a probability *proba_leave*
- the agent chooses as new target a random location in a random building

```
model my_model

global {
}

species my_species {
}

experiment my_model type: gui {
}
```

Answer:

```
species inhabitant skills: [moving]{
   //definition of variables

   reflex leave when: (target = nil) and (flip(proba_leave)) {
      target <- any_location_in(one_of(building));
   }

   //aspect
}
```

inhabitant species: *move* reflex

TODO: define a new reflex (named *move*) for the *inhabitant* species:

- activated when the agent has a target
- the agent moves on the network toward its target
- when it reaches its target, it drops its target.

```
model my_model

global {
}

species my_species {
}

experiment my_model type: gui {
}
```

Answer:

```
species inhabitant skills: [moving]{
    //definition of variables and reflex

    reflex move when: target != nil {
        do goto target: target on: road_network;
        if (location = target) {
            target <- nil;
        }
    }
    //aspect
}
```

The **goto** action can only be used by inhabitant agents because they have the moving skill.

The **goto** action takes into account the ***speed*** built-in inhabitant variable and the ***step*** global variable.

Modification of the simulation step duration

Remark: agents move very slow (more precisely, in 1 simulation step, they move a short distance)

TODO: set the duration of 1 simulation step to 10 seconds.

```
model my_model

global {
}

species my_species {
}

experiment my_model type: gui {
}
```

Answer:

```
global {
    //definition of global variables
    float step <- 1#mn;
    ...
}
```

The symbol # can also be used for units (e.g. **#min**, **#m**, **#h** …)

step is a global built-in variable that represents the duration of 1 simulation step (default value = 1 s)

Similarly **cycle** is a global built-in variable that contains the number steps for the simulation beginning.

End of step 3

Now let take congestion into account!

Step 4: introduction of congestion

Objectives:

- Make the roads “aware” of the state in terms of congestion

- Addition of a new reflex to update speed on roads

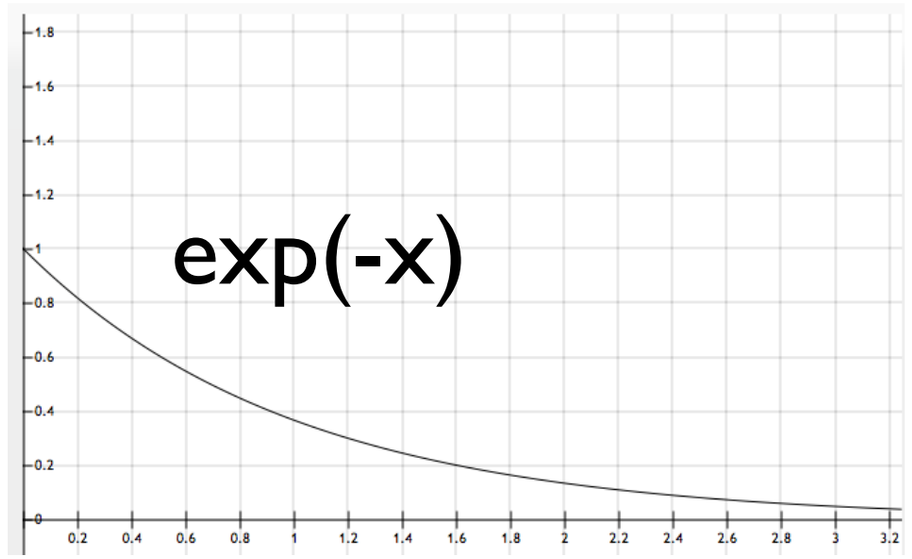
road species: new dynamic variables

```
model my_model

global {
}

species my_species {
}

experiment my_model type: gui {
}
```

TODO: define 3 new dynamic variables for road agents

- capacity (type = float, init value = 1 + road perimeter/30.0)
- nb_drivers (type = int, update = number of inhabitants at a distance of 1 #m)
- speed_rate (type = float, update = exp(-nb_drivers/capacity)

Modify the geom aspect in order to add a buffer of size 3* speed_rate around the geometry and change its color in red.



Answer:

```
species road {
    float capacity <- 1 + shape.perimeter/30;
    int nb_drivers <- 0 update: length(inhabitant at_distance 1);
    float speed_rate <- 1.0 update:  exp(-nb_drivers/capacity);
    aspect geom {
        draw (shape + 3 * speed_rate) color: #red;
    }
}
```

global block: definition of the *update_speed* reflex

TODO: define a new global reflex (update_speed) that associates to each road a weight (function of the speed_rate) in a map data structure. It then updates the weight of the graph edge with this map.

```
model my_model

global {
}

species my_species {
}

experiment my_model type: gui {
}
```

Answer:

```
global {
    //variables and init
    reflex update_speed {
        map<road,float> new_weights <- road as_map (each::each.shape.perimeter * each.speed_rate);
        road_network <- road_network with_weights new_weights;
    }
}
```

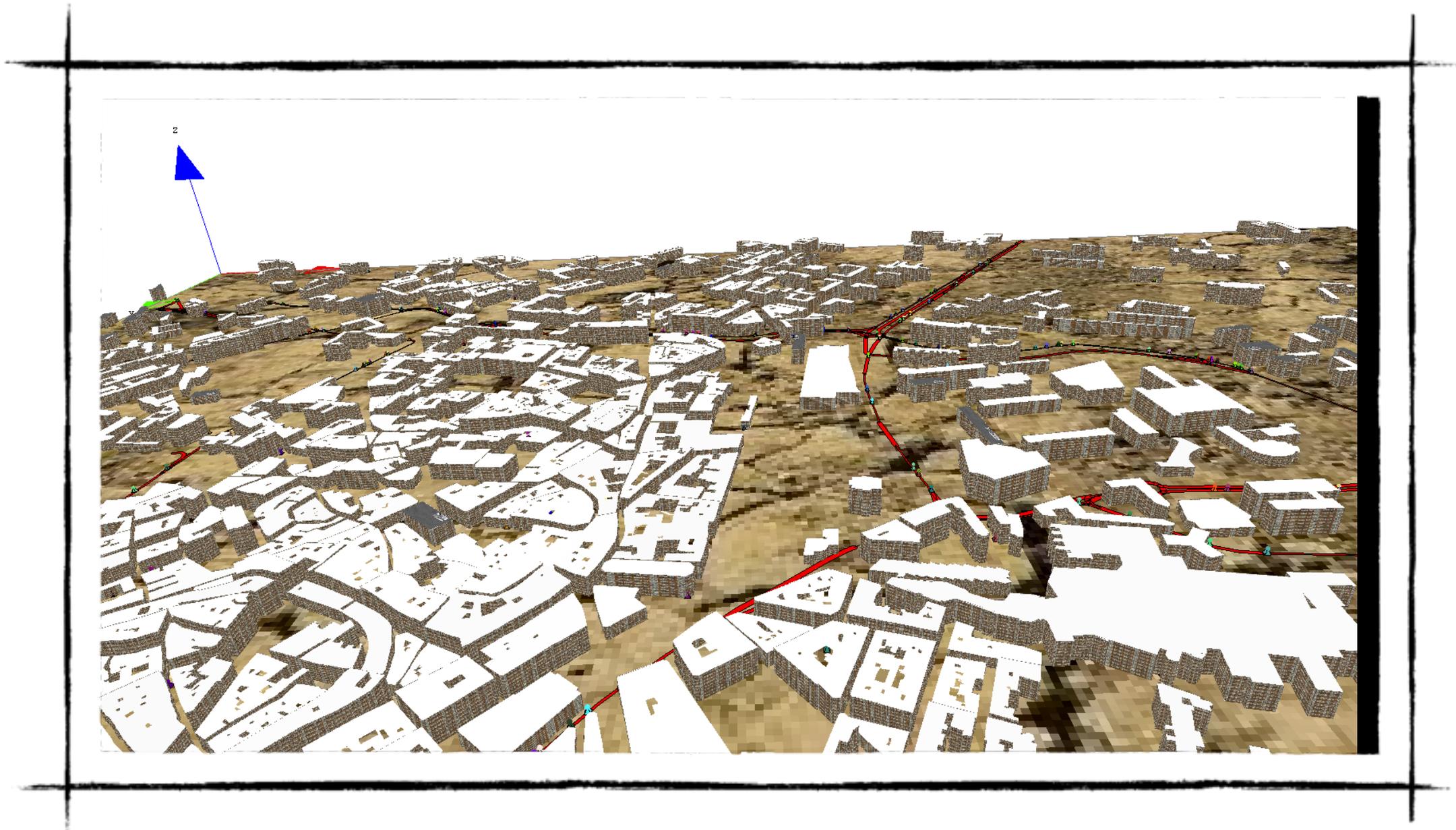
End of step 4

Now, let's have a nice visualisation.

Step 5: definition of a 3D display

Objectives:

- Definition of 3D aspects for building and inhabitant agents

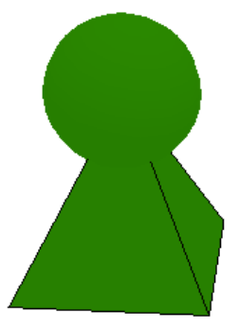


inhabitant species: aspect threeD

TODO: define a new threeD aspect that:

- draw a pyramid with a height of 5m and a color color
- draw a sphere (with a radius of 2m) at a height of 5m and with a color color.

Answer:



```
model my_model

global {
}

species my_species {
}

experiment my_model type: gui {
}
```

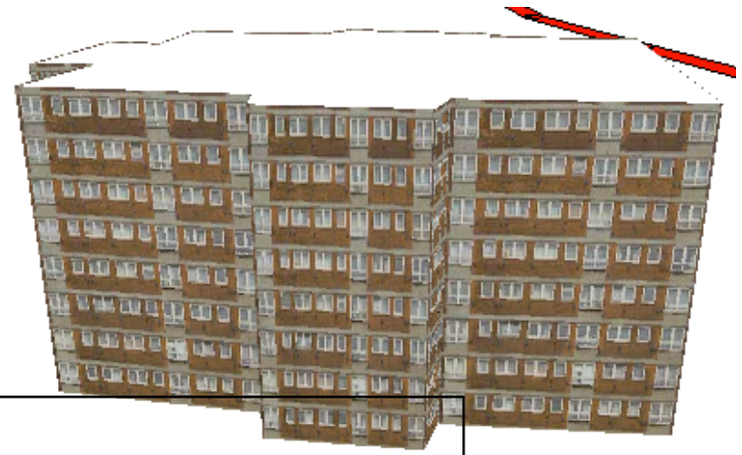
```
species inhabitant skills: [moving]{
    //definition of the variables, reflex and aspect

    aspect threeD{
        draw pyramid(5) color: color;
        draw sphere(2) at: location + {0,0,5} color: color;
    }
}
```

building species: improve the display

```
model my_model

global {
}

species my_species {
}

experiment my_model type: gui {
}
```

TODO:

- add a height variable (type = int) to the building species, with a value read from the shapefile.
- add a new aspect (threeD) drawing the shape of the building with a height (height) and a texture.



Answer:

```
global {
    init {
        create building from: shapefile_buildings with:[height::int(read("height"))];
```

read attribute value and store it into an attribute.

```
species building {
    int height;
    aspect threeD {
        draw shape color: #gray depth: height texture: ["../includes/roof_top.png","../includes/texture5.jpg"];
    }
}
```

Display of the agents

TODO: Add in the map display a background picture and modify the aspect of building and inhabitants (using threeD).
Use the *opengl* mode for the display.

```
model my_model

global {
}

species my_species {
}

experiment my_model type: gui {
}
```

Answer:

The use of **type: opengl** is mandatory to display 3D in a **display**.
It can also be used for 2D simulation (and often makes the zoom in/out smoother)

```
experiment traffic type: gui {
    output {
        display map type: opengl{
            image "../includes/soil.jpg" refresh: false;
            species building aspect: threeD refresh: false;
            species road aspect: geom ;
            species inhabitant aspect: threeD;
        }
    }
}
```

End of step 5