GAMA Platform: introduce heterogeneity in the environment with ChouChevLoup model

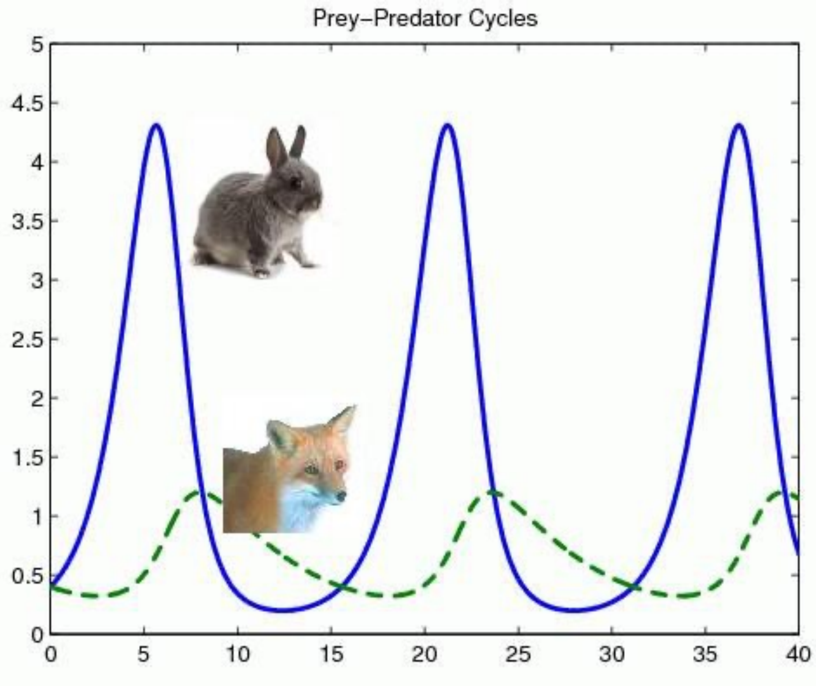
Benoit GAUDOU, *University Toulouse 1 Capitole, IRIT; benoit.gaudou@gmail.com*

CC BY NC SA

The Lotka-Volterra model (prey-predator model)

- This model represents the population dynamics of 2 species interacting, 1 being prey and the other one the predator.

$$\boxed{\begin{aligned}\frac{dx}{dt} &= \alpha x - \beta xy \\ \frac{dy}{dt} &= \delta xy - \gamma y\end{aligned}}$$



The prey-predator model

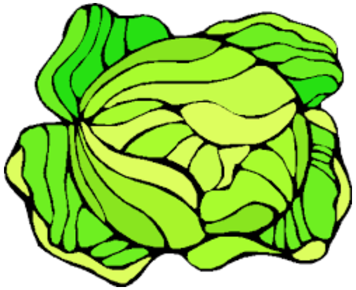
- We consider a system with prey (goat) and predator (wolf) animals.
- Animals move randomly in a space.
- Predators can hunt and kill prey. Prey can eat some cabbages on the ground.
- Both preys and predators can reproduce.
- Both preys and predators can die from natural reasons.

The ChouChevLoup model

- The ChouChevLoup model is a Prey-Predator model with a Cabbage ressource, where Preys are goats and predators are wolves.

- Preys and Predators are located in a discret space (a grid).
- Cabbages ressources grow on the pixels.
  Pixels can contain a maximum quantity of cabbage (named **carrying capacity**). The quantity of the cabbage is represented by its **biomass**.
- Goats eat cabbages.

Model 1: The cabbages

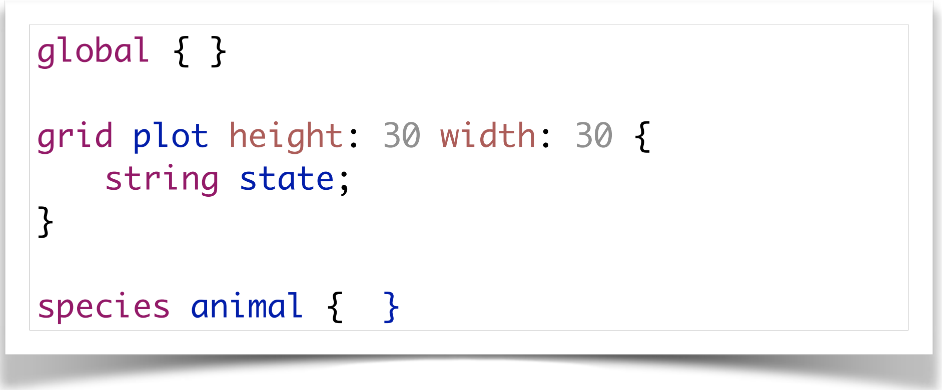


- A landscape made up of 900 square spatial units covered with a wild cabbage species.
- The **carrying capacity** of the environment is a random value between 0 and 10 **biomass**.
- Initially, the cabbages biomass is random between 0 and the local carrying capacity.
- Wild cabbage biomass grows with a logistic function with a growth rate equal to 0,2.
- Display the maps of biomass and of carrying capacity.

$$\mathbf{X(t+1)} = \mathbf{X(t)} * \left(\mathbf{1} + \mathbf{growth\_rate} * \left(\mathbf{1} - \frac{\mathbf{X(t)}}{\mathbf{carrying\_capacity}}\right)\right)$$

Notes on the model.

- Every kind of agent has **built-in attributes**:
  - **name** (a string)
  - **shape** (a geometry) (default value = a point)
  - **location** (a point) (value = the centroid of its shape)
- In addition, **grid** agents have additional built-in attributes:
  - **grid_x** (an integer)
  - **grid_y** (an integer)
  - **color** (a color)
  - **grid_value** (used when grid is created from a data file)
  - **neighbors** (list of plot at a distance 1)

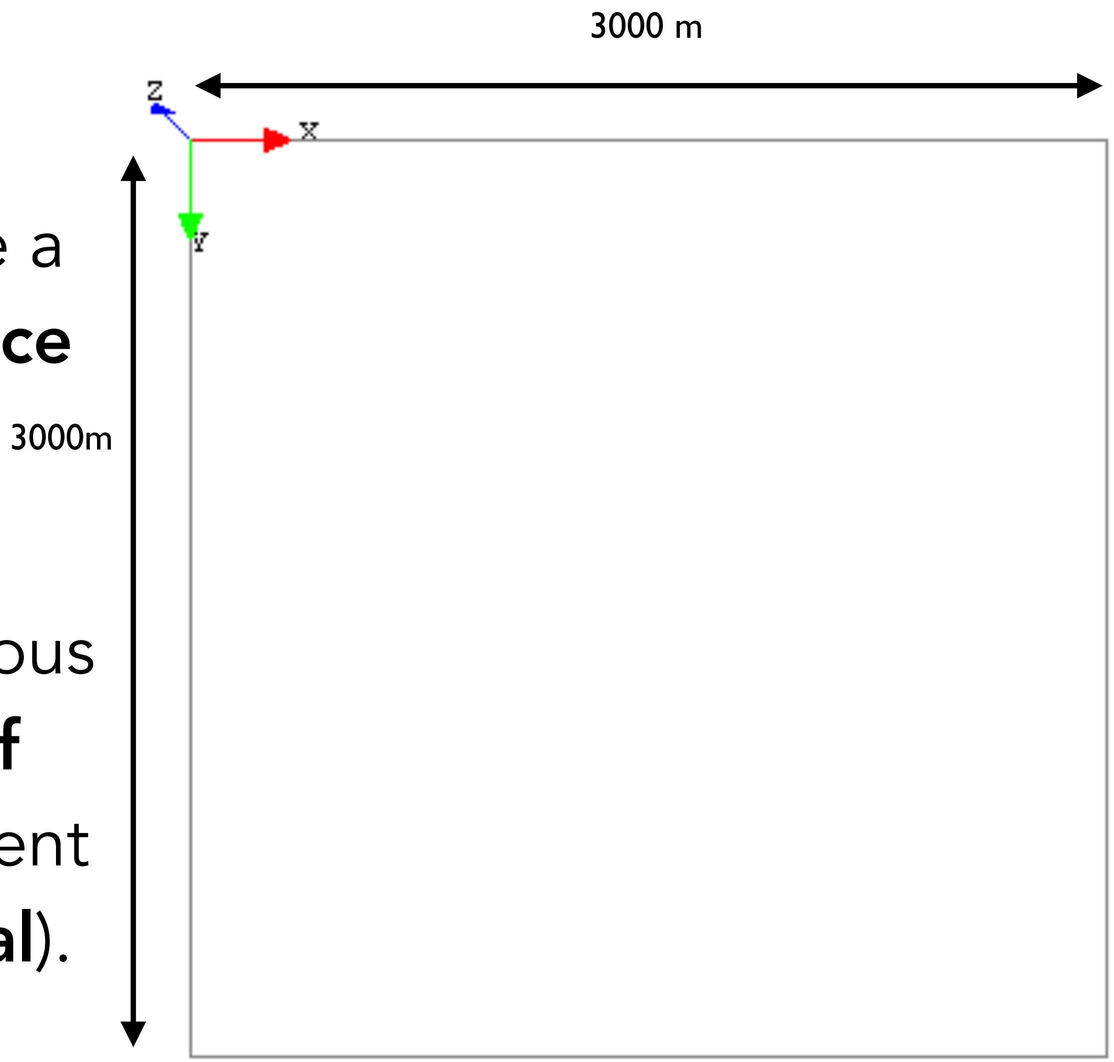


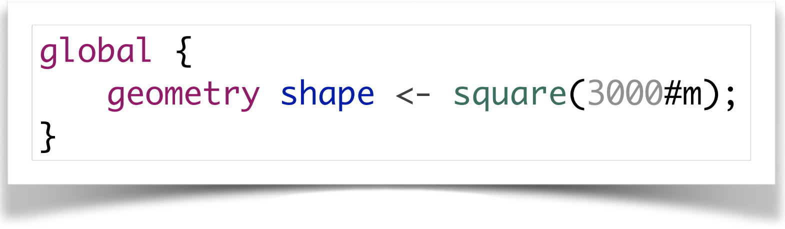
```
global { }

grid plot height: 30 width: 30 {
    string state;
}

species animal {  }
```

Space in GAMA

- In GAMA, agents have a **location** in a **reference continuous space**.

- The reference continuous space is the **shape of the world** (single agent instance of the global).



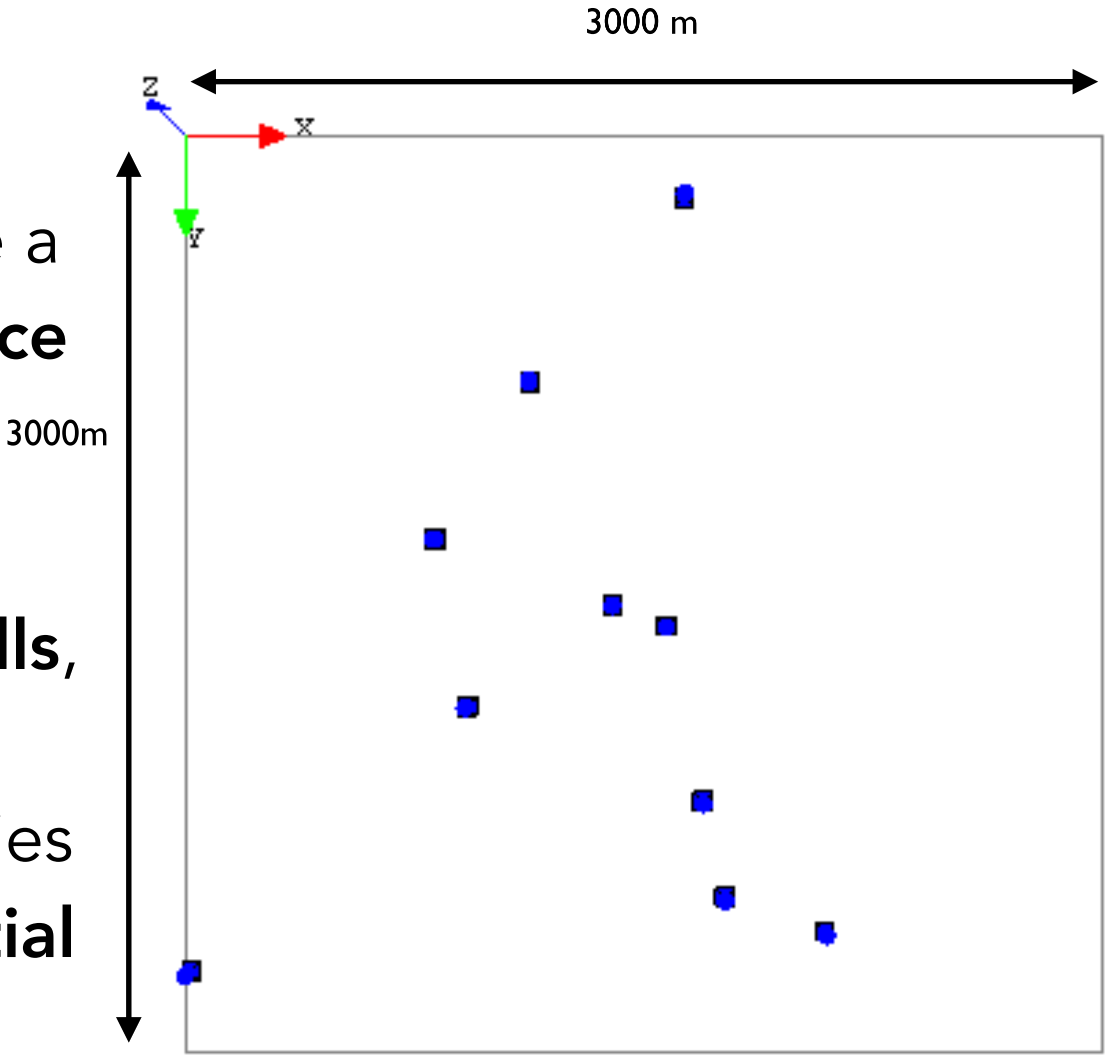


```
global {
    geometry shape <- square(3000#m);
}
```

Space in GAMA

- In GAMA, agents have a **location** in a **reference continuous space**.

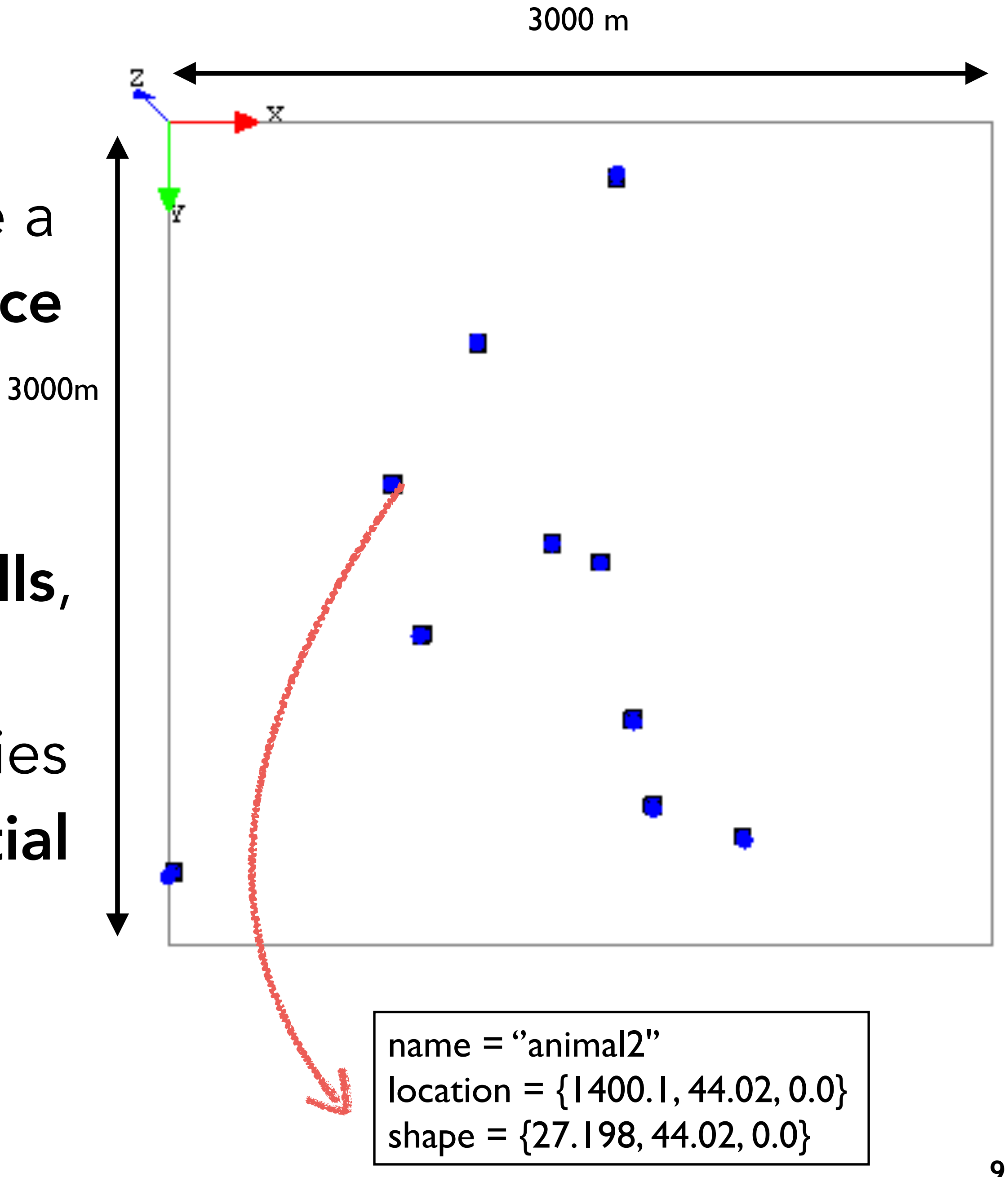
- To **create a grid of cells**, we need to create explicitly a new species with a particular **spatial organisation** (a particular topology).



Space in GAMA

- In GAMA, agents have a **location** in a **reference continuous space**.

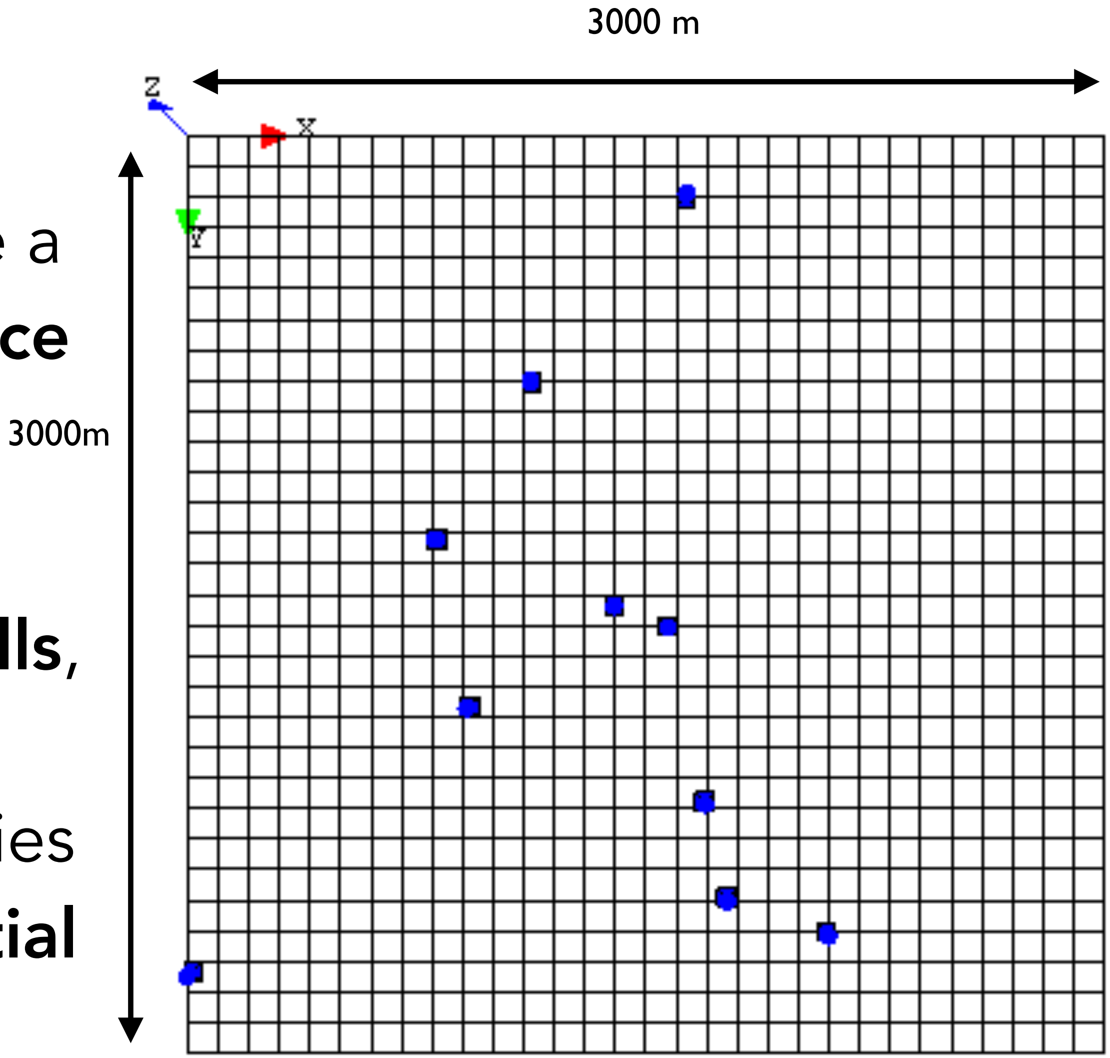
- To **create a grid of cells**, we need to create explicitly a new species with a particular **spatial organisation** (a particular topology).



name = "animal2"
location = {1400.1, 44.02, 0.0}
shape = {27.198, 44.02, 0.0}

Space in GAMA

- In GAMA, agents have a **location** in a **reference continuous space**.
- To **create a grid of cells**, we need to create explicitly a new species with a particular **spatial organisation** (a particular topology).

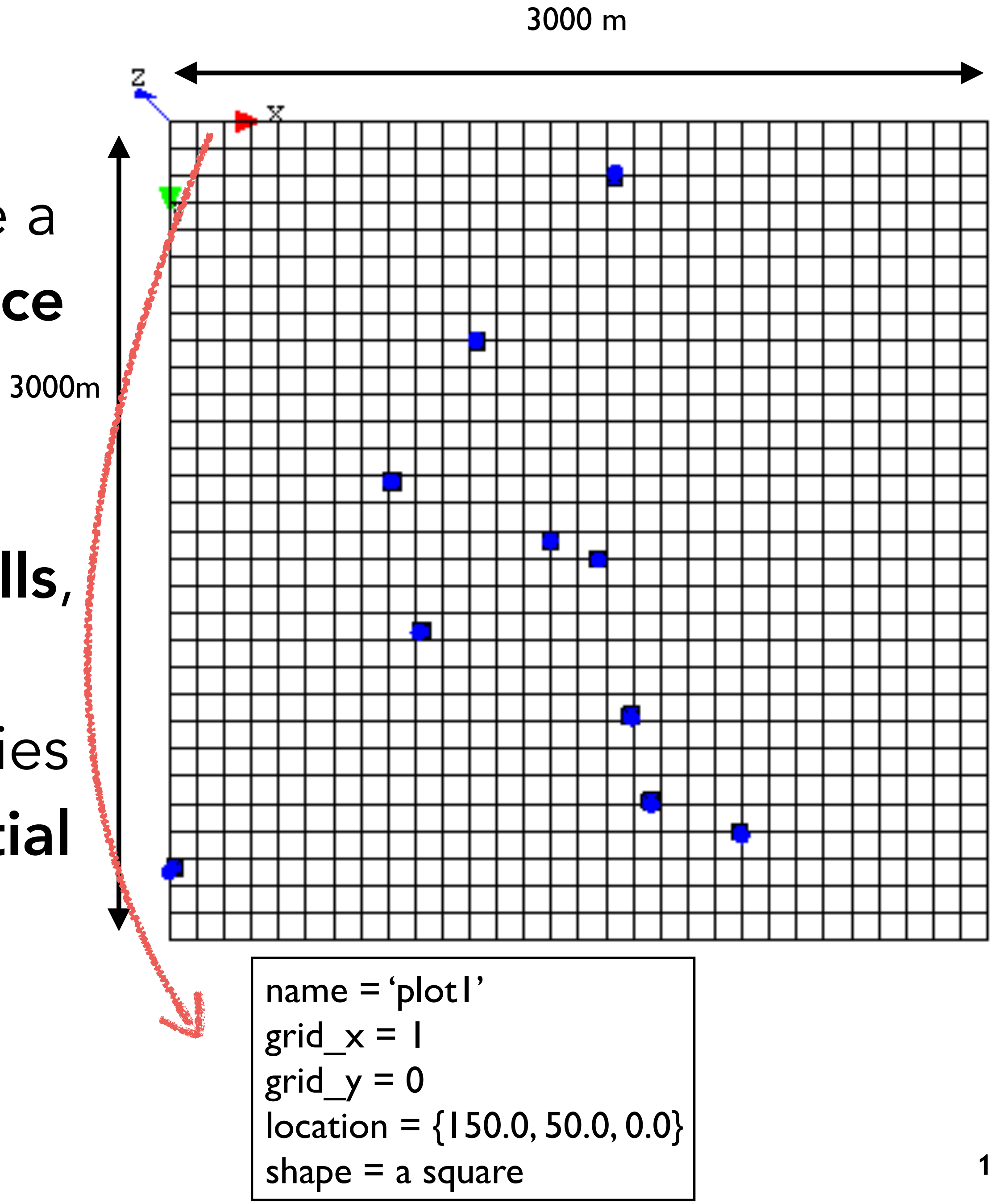


Addition of a 30x30 grid

```
grid plot height:30 width:30 {
}
```

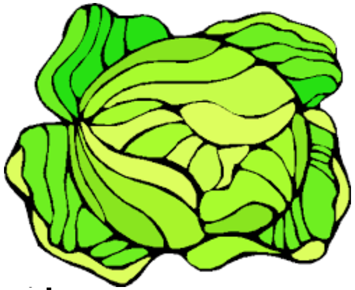
Space in GAMA

- In GAMA, agents have a **location** in a **reference continuous space**.
- To **create a grid of cells**, we need to create explicitly a new species with a particular **spatial organisation** (a particular topology).



name = 'plot1'
grid_x = 1
grid_y = 0
location = {150.0, 50.0, 0.0}
shape = a square

The cabbages



- A landscape made up of 900 square spatial units covered with a wild cabbage species.
- The carrying capacity of the environment is a random value between 10 and 0 biomass.
- Initially, the cabbages biomass is random between 0 and the local carrying capacity.
- Wild cabbage biomass grows with a logistic function with a growth rate equal to 0,2.
- Display the maps of biomass and of carrying capacity.

$$\mathbf{X}(\mathbf{t}+\mathbf{1}) = \mathbf{X}(\mathbf{t}) * \left(\mathbf{1} + \mathbf{growth\_rate} * \left(\mathbf{1} - \frac{\mathbf{X}(\mathbf{t})}{\mathbf{carrying\_capacity}}\right)\right)$$

Use of a grid topology

- Agents can be organised following 3 topologies (continuous, grid or graph).
- The grid statement allows modeler to define a **species of agents** organised as a grid.
  - they have a square shape
  - they have **additional built-in attributes** :
    - grid_x, grid_y : coordinates in the grid
    - neighbors : list of neighbours at a distance 1
    - grid_value : initialised when the grid has been created from an .asc file.
- **Agents in a grid are created automatically.**

```
grid plot height: grid_size width: grid_size neighbors: 8 {
    // attributes
    // init
    // reflexes
    // aspects
}
```

The number of neighbors: can be 4, 6 or 8

dimension of the grid

Display of grid agents

- **grid agents have a built-in aspect :**
  - a square/hexagon with the built-in attribute `color` as color.
  - To display grid agents using this built-in aspect:

```
display biomass {
    grid plot lines: #black;
}
```

Use the grid statement in a display to use the built-in display

- But additional aspects can be defined and used.

```
grid plot height: grid_size width: grid_size neighbors: 6 {
    aspect plotCarryingCapacity {
        draw square(1) color: rgb(0,255*carrying_capacity/max_carrying_capacity,0);
    }
}
```

```
    display carryingCapacity {
        species plot aspect: plotCarryingCapacity;
    }
```

They are displayed as any other species

A landscape made up of 900 square spatial units covered with a wild cabbage species.

➡ Define a species of agents (organised as a grid),

- with 2 attributes related to cabbages:

  biomass of cabbages and the carrying_capacity.

- with an attribute to compute the color.

```
grid plot height: 30 width: 30 neighbors: 8 {

    float biomass;
    float carrying_capacity;

    rgb color <- rgb(0,255*biomass/max_carrying_capacity,0)
        update: rgb(0,255*biomass/max_carrying_capacity,0);

}
```

Initialisation of agents attributes

- ➡ The carrying capacity of the environment is equal to a random value from 0 to 10 biomass units.
- ➡ Initially, the cabbages biomass is random between 0 and the local carrying capacity.

We define a global variable for this carrying capacity max.

```
global {

    float max_carrying_capacity <- 10.0;
}

grid plot height: 30 width: 30 neighbors: 8 {
    init {
        carrying_capacity <- rnd(max_carrying_capacity);
        biomass <- rnd(carrying_capacity);
        color <-  rgb(0,255*biomass/max_carrying_capacity,0);
    }
}
```

a color is defined by its red, green and blue components (a number betweenn 0 and 255)

- ➡ Display it

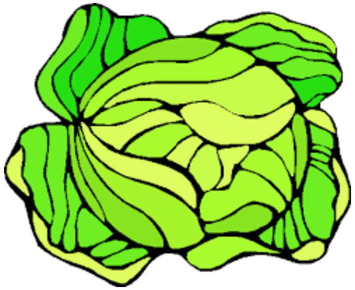
Wild cabbage biomass grows with a logistic function with a growth rate equal to 0,2.

- ➡ requiers to define a reflex for plots.
- ➡The logistic function is the following one:

$$X(t+1) = X(t) * (1 + growth\_rate * (1 - \frac{X(t)}{carry\_capacity}))$$

```
global {
     float growth_rate <- 0.2 ;
}

grid plot height: 30 width: 30 neighbors: 8 {

     reflex grow {
          if(carrying_capacity != 0){
               biomass <- biomass * (1 + growth_rate * (1 - biomass/carrying_capacity));
          }
     }
}
```

The cabbages



- A landscape made up of 900 square spatial units covered with a wild cabbage species.
- The carrying capacity of the environment is a random value between 10 and 0 biomass.
- Initially, the cabbages biomass is random between 0 and the local carrying capacity.
- Wild cabbage biomass grows with a logistic function with a growth rate equal to 0,2.
- Display the maps of biomass and of carrying capacity.

```
experiment e {
     output {
          display biomass {
               grid plot lines: #black;
          }
          display carrying_capacity {
               species plot aspect: carry;
          }
     }
}
```

Model 2: Introduction of wolves and goats

- We want to add wolves and goats in the model. They will be located on the center of a plot.
- **Wolves**:
  - number : 3
  - aspect : red circle
- **Goats**:
  - number : 10
  - aspect : blue circle

Model 2: Introduction of wolves and goats

- We want to add wolves and goats in the model. They will be located on the center of a plot.
- **Wolves**:
  - number : 3
  - aspect : red circle
- **Goats**
  - number : 10
  - aspect : blue circle
- **Create 2 species, with 1 aspect.**
- **Create agents of these species in the init from the global.**

Model 2: Introduction of wolves and goats

- Create 2 species, with 1 aspect.
- Create agents of these species in the init from the global.
- Display them!

```
global {
    init {
        create goat number: 3;
        create wolf number: 10;
    }
}

species wolf {
    aspect redCircle {
        draw circle(1) color: #red;
    }
}

species goat {
    aspect blueSquare {
        draw square(1) color: #blue;
    }
}
```

```
experiment cabbagesExp type: gui {
	output {
		display biomass {
			grid plot lines: #black;
			species wolf aspect: redCircle;
			species goat aspect: blueSquare;
		}
	}
}
```

Issue: goats and wolves are not located at the center of plots

- The location should be set at the center of a plot.

- Solution: when a goat/wolf is created, choose a plot and set the goat/wolf location at the center of the plot.

```
species wolf {
    init {
        location <- one_of(plot).location;
    }
}

species goat {
    init {
        location <- one_of(plot).location;
    }
}
```

The name of the species can be used as the list of all agents of the species

operator that choose a random element of a list/ species

Issue 2: nothing is done to avoid to have 2 animals on the same plot.

- The plot should "know" if an animal is on it.
- **Solution**: add an attribute to store if the plot is free or not. We also store the plot in the animal.

```
species wolf {
    plot my_plot;
    init {
        my_plot <- one_of(plot where (each.is_free = true));
        location <- my_plot.location;
        my_plot.is_free <- false;
    }

    aspect redCircle {
        draw circle(1) color: #red;
    }
}
```

where operator allows to return the set of agents/elements of a container that fulfil a condition

Model 3: Make wolves and goats move

- The goats and wolves move at each step on a neighbor free plot

- Add a reflex to goat/wolf to move:
  - choose a plot in the neighbourhood of the current plot
  - move on it
  - free the previous plot

Model 3: Make wolves and goats move

- Add a reflex to goat/wolf to move:
  - choose a plot in the neighbourhood of the current plot
  - move on it
  - free the previous plot

grid agents have a built-in neighbours attribute storing the agents at a distance 1

```
species wolf {
    plot my_plot;

    reflex move {
        plot next_plot <- one_of(my_plot.neighbors where(each.is_free = true));

        my_plot.is_free <- true;
        next_plot.is_free <- false;

        my_plot <- next_plot;
        location <- next_plot.location;
    }
```

Move = set is_free attributes of the old and new my_plot. Move (= change the location) of the agent to the new_plot

Model 3: Make wolves and goats move

```
species wolf {
    plot my_plot;

    reflex move {
        plot next_plot <- one_of(my_plot.neighbors where(each.is_free = true));

        my_plot.is_free <- true;
        next_plot.is_free <- false;

        my_plot <- next_plot;
        location <- next_plot.location;
    }
```

This piece of code is used in init and move reflex. **Let's create an action**, that can be used in both cases.

Model 3: Make wolves and goats move

```
species wolf {
    plot my_plot;

    init {
        plot random_plot <- one_of(plot where (each.is_free = true));
        do move_to_cell(random_plot);
    }

    reflex move {
        plot next_plot <- one_of(my_plot.neighbors where(each.is_free = true));
        do move_to_cell(next_plot);
    }

    action move_to_cell(plot new_plot) {
        if(my_plot != nil) {
            my_plot.is_free <- true;
        }
        new_plot.is_free <- false;
        my_plot <- new_plot;
        location <- new_plot.location;
    }
}
```

This piece of code is used in init and move reflex. Let's create an action, that can be used in both cases.

Notes: goat and wolf agents are very similar!

```
species goat {
    plot my_plot;
    init {
      my_plot <- one_of(plot where (each.is_free
= true));
      location <- my_plot.location;
      my_plot.is_free <- false;
    }

    reflex move {
      plot next_plot <- one_of(my_plot.neighbors
where(each.is_free = true));
      my_plot.is_free <- true;
      next_plot.is_free <- false;
      my_plot <- next_plot;
      location <- next_plot.location;
    }

    aspect blueSquare {
      draw square(2) color: #blue;
    }
}
```

```
species wolf {
    plot my_plot;
    init {
      my_plot <- one_of(plot where (each.is_free =
true));
      location <- my_plot.location;
      my_plot.is_free <- false;
    }

    reflex move {
      plot next_plot <- one_of(my_plot.neighbors
where(each.is_free = true));
      my_plot.is_free <- true;
      next_plot.is_free <- false;
      my_plot <- next_plot;
      location <- next_plot.location;
    }

    aspect redCircle {
      draw circle(1) color: #red;
    }
}
```

- goat and wolf are 2 kinds of animals which share a lot of attributes and behaviours => introduction of a new more general species

Introduction of the species animal. wolf and goat inherit from it.

```
species animal {
      plot my_plot;
      init {
            my_plot <- one_of(plot where (each.is_free = true));
            location <- my_plot.location;
            my_plot.is_free <- false;
      }

      reflex move {
            plot next_plot <- one_of(my_plot.neighbors where(each.is_free = true));
            my_plot.is_free <- true;
            next_plot.is_free <- false;
            my_plot <- next_plot;
            location <- next_plot.location;
      }
}

species wolf parent: animal {
      aspect redCircle {
            draw circle(1) color: #red;
      }
}

species goat parent: animal {
      aspect blueSquare {
            draw square(2) color: #blue;
      }
}
```

wolf inherits from animal:
it gets attributes, init and reflex from animal.
But it can have its own attributs and behaviours

Model 4: Make wolves and goats die…

- Wolves and goats can die (for natural reasons)

- We represent that using an energy amount to animals. This energy decreases at each step. When the energy reaches 0, the animal dies.

- The energy management will be the same for wolves and goats, **so it can be defined at the animal level.**

Addition of energy, its decrease step by step and its effect on animal life.

```
species animal {
    float energy <- initial_energy;

    // Other reflexes

    reflex energy_loss {
        energy <- energy - 1;
    }

    reflex death when: energy <= 0.0 {
        do die;
    }

}
```

The new attribute

New reflex to decrease energy at each simulation step.
Note: we could replace this reflex by a update in the energy declaration

Built-in action, to make the agent die.

Model 5: goats can get energy by eating cabbages

- Goats can eat a given amount of cabbages from the plot on which they are located.
  This cabbages are transformed into energy.

```
global {
    float max_cabbages_eat <- 2.0;
}

species goat parent: animal {

    reflex eat_cabbage {
        float cab <- min([max_cabbages_eat, my_plot.biomass]);
        energy <- energy + cab;
        my_plot.biomass <- my_plot.biomass - cab;
    }
// ...
```

The maximum of cabbages that can be eaten.

A goat cannot take more than the biomass in a plot.

Note: reflex in goat are executed before the ones in the animal (more specific first)

Model 6: reproduction of the animals

- **TODO**: when animals reach a certain amount of energy they can "reproduce":

- an animal produces a new animal of the same species in a neighbour free plot;
- its energy is shared with its child.

```
global {
	float reproduction_threshold <- 20.0;
}

species animal {

	reflex reproduce when: energy >= reproduction_threshold {
		plot plot_for_child <- one_of(my_plot.neighbors where(each.is_free = true));

		if(plot_for_child != nil) {
			create species(self) number: 1 {
				do move_to_cell(plot_for_child);
				self.energy <- myself.energy / 2;
			}
			energy <- energy / 2;
		}
	}
```

species(self) returns the species of the current agent (i.e. either wolf or goat). This allows to have the same code for both kinds of animal.

self refers to the current agent (here the new created agent, the child), whereas **myself** refers to the agent that has called the create (the parent agent)

Note: the init of the species is called before the create block.

Model 7: Make wolves “hunt” and eat goats

- The wolves will attempt to eat goat around it.

- Update the wolf move reflex:
  - look for goats in its neighbourhood,
  - if no goat
    - choose a random next_plot
  - if there is some plots with goats on them
    - choose one of them randomly
    - take its energy
    - kill the goat on it (ask it to die)
    - move on this plot

Algorithm to make the wolves “hunt”

```
reflex move {
     plot next_plot <- nil;

     list<plot> neigh <- my_plot.neighbors where(!empty(goat inside each ));
     if(empty(neigh)) {
          next_plot <- one_of(my_plot.neighbors where(each.is_free = true));
     } else {
          next_plot <- one_of(neigh);
          goat victim <- one_of(goat inside next_plot);
          energy <- energy + victim.energy;
          ask victim {
               write "" + self + " will die";
               do die;
          }
     }

     do move_to_cell(next_plot);
}
```

Spatial query to get the goat agents inside a plot (each)

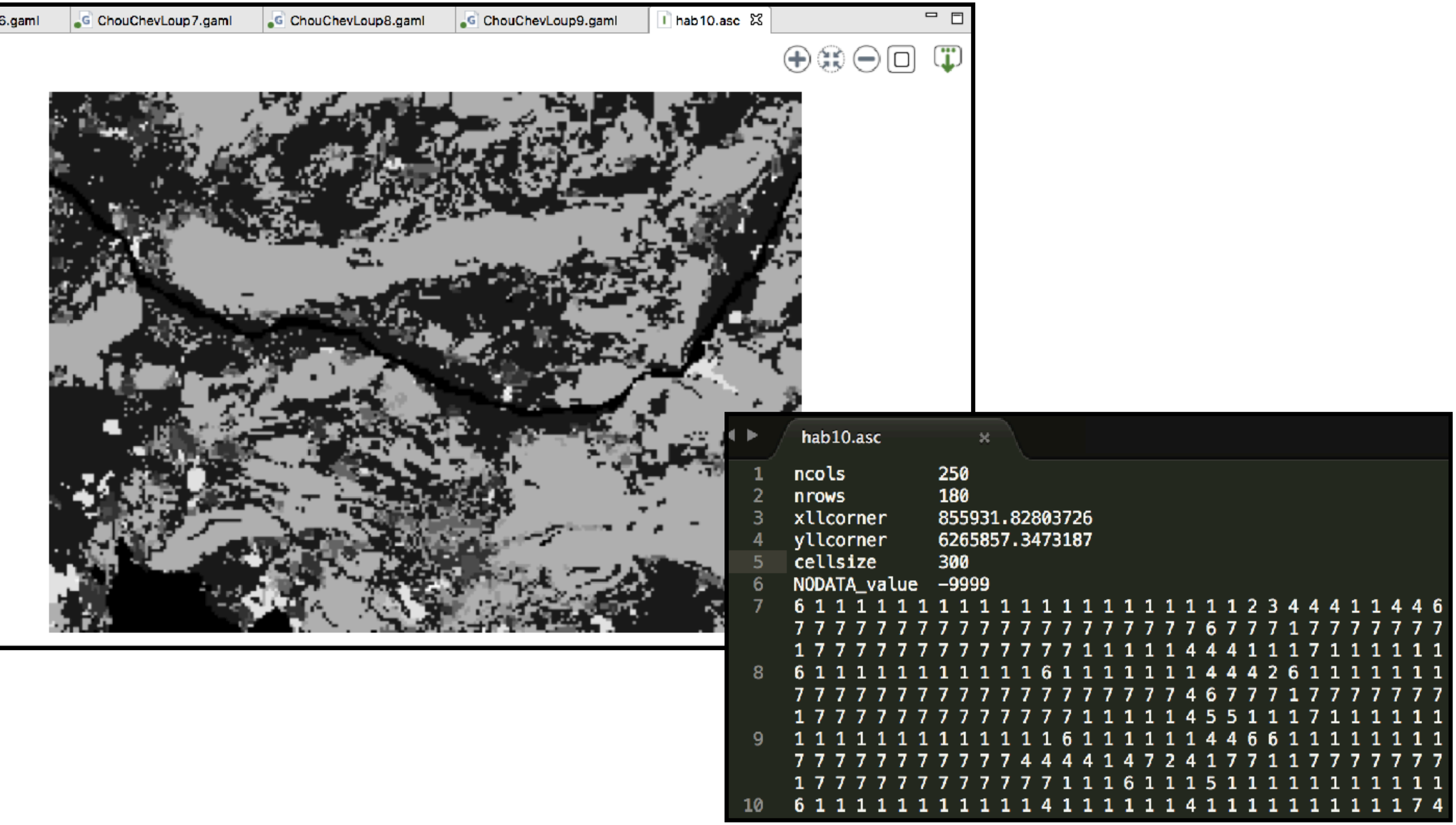
Kill an agent removes it from the simulation

Note: **reflex move** in wolf overwrites the **reflex move** in animal. Only the one of wolf will be executed by wolf agents.

Model 8: add a chart to observe the evolution of both populations

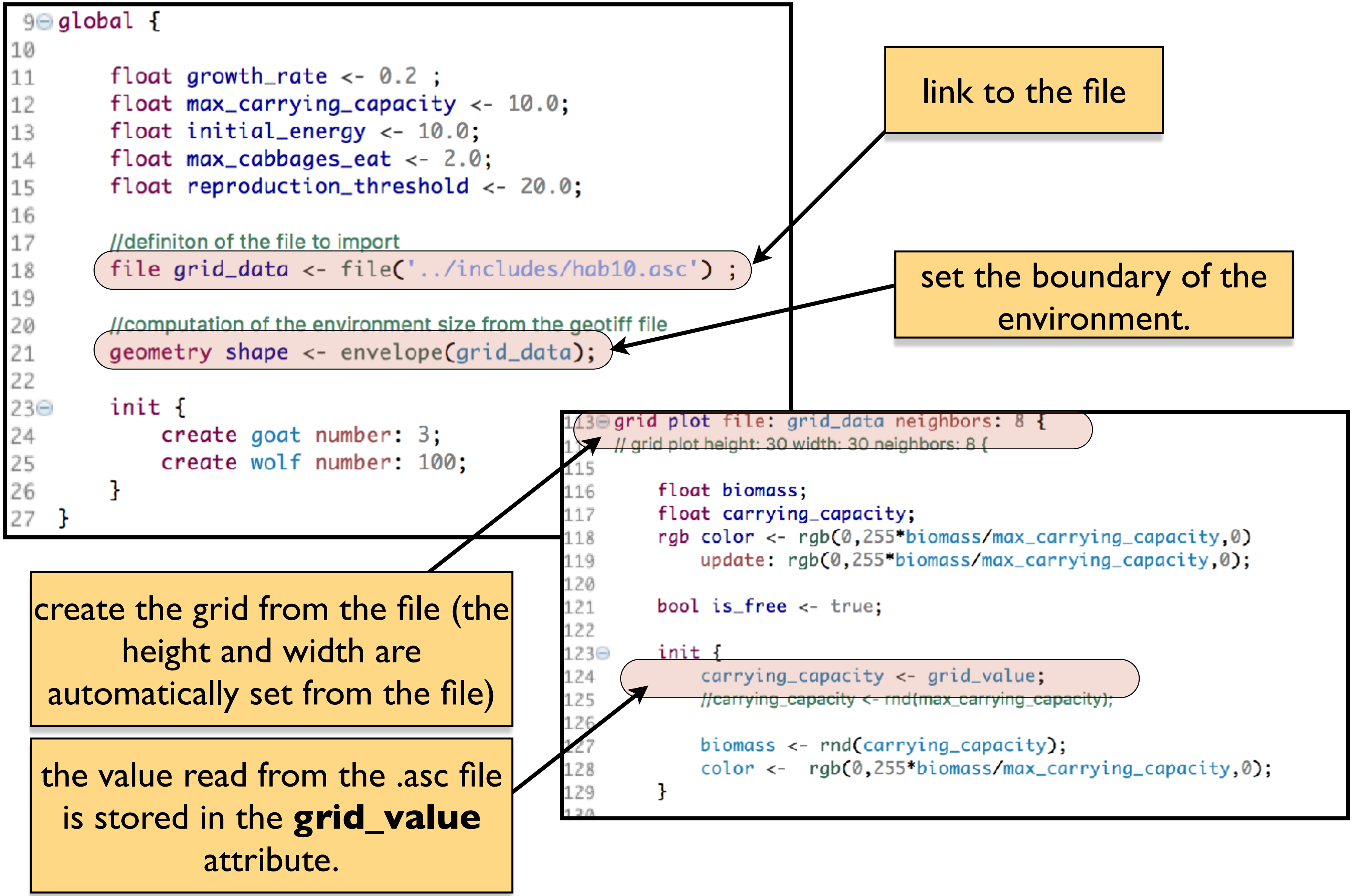
```
experiment cabbagesExp type: gui {
    output {
        display biomass {
            grid plot lines: #black;
            species wolf aspect: redCircle;
            species goat aspect: blueSquare;
        }

        display plots {
            chart "Nb animals" type: series {
                data "#wolves" value: length(wolf);
                data "#goats" value: length(goat);
            }
        }

    }
}
```

Model 9: initialise the environment from an .asc file.



```
ncols         250
nrows         180
xllcorner     855931.82803726
yllcorner     6265857.3473187
cellsize      300
NODATA_value  -9999
6 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 3 4 4 4 1 1 4 4 6
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 6 7 7 7 1 7 7 7 7 7 7 7
1 7 7 7 7 7 7 7 7 7 7 7 7 7 1 1 1 1 1 4 4 4 1 1 1 7 1 1 1 1 1 1
6 1 1 1 1 1 1 1 1 1 1 1 6 1 1 1 1 1 1 1 4 4 4 2 6 1 1 1 1 1 1 1
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 4 6 7 7 7 1 7 7 7 7 7 7 7
1 7 7 7 7 7 7 7 7 7 7 7 7 7 1 1 1 1 1 4 5 5 1 1 1 7 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 6 1 1 1 1 1 1 4 4 6 6 1 1 1 1 1 1 1 1
7 7 7 7 7 7 7 7 7 7 7 4 4 4 4 1 4 7 2 4 1 7 7 1 1 7 7 7 7 7 7 7
1 7 7 7 7 7 7 7 7 7 7 7 7 1 1 1 6 1 1 1 5 1 1 1 1 1 1 1 1 1 1 1
6 1 1 1 1 1 1 1 1 1 1 1 4 1 1 1 1 1 1 4 1 1 1 1 1 1 1 1 1 1 7 4
```

Model 9: initialise the environment from an .asc file.



```
global {

	float growth_rate <- 0.2 ;
	float max_carrying_capacity <- 10.0;
	float initial_energy <- 10.0;
	float max_cabbages_eat <- 2.0;
	float reproduction_threshold <- 20.0;

	//definiton of the file to import
	file grid_data <- file('../includes/hab10.asc') ;

	//computation of the environment size from the geotiff file
	geometry shape <- envelope(grid_data);

	init {
		create goat number: 3;
		create wolf number: 100;
	}
}
```

link to the file

set the boundary of the environment.

```
grid plot file: grid_data neighbors: 8 {
// grid plot height: 30 width: 30 neighbors: 8 {

	float biomass;
	float carrying_capacity;
	rgb color <- rgb(0,255*biomass/max_carrying_capacity,0)
		update: rgb(0,255*biomass/max_carrying_capacity,0);

	bool is_free <- true;

	init {
		carrying_capacity <- grid_value;
		//carrying_capacity <- rnd(max_carrying_capacity);

		biomass <- rnd(carrying_capacity);
		color <-  rgb(0,255*biomass/max_carrying_capacity,0);
	}
```

create the grid from the file (the height and width are automatically set from the file)

the value read from the .asc file is stored in the **grid_value** attribute.

Plot everything !

- Evolution of the biomass
- Evolution of the number of goats, wolves
- Evolution of the average energy
- Evolution of the average harvest rate
- …

Potential improvements

- Plot can diffuse biomass in their neighborhood
- Goats looking for plots with more biomass
- Goats moving away from wolves
- Goats alerting the others when they see a wolf
- Goats having a chance to escape the wolves
- Goat's offspring inherits harvest rate from genitor +/- delta
- Wolves resting after having eaten a goat
- Wolves hunting together and sharing the goat