Spatial Simulation

Winter Semester 2023 / 24, MSc Applied Geoinformatics

Interaction in geographic space

Create agents from a geospatial file

```
global {
    // 1) Load geospatial file
    file pasture_file <- file("../includes/pasture.geojson");
    // 2) Define the extent of the study area
    geometry shape <- envelope(pasture_file);
    geometry pasture_polygon <- geometry(pasture_file);

    // 3) Create the agents
    init {
        create cows number: 6{
          location <- any_location_in(pasture_polygon);
       }
    }
}
```

The geometry variable **shape**

When working with GIS data, we need to define the extent of our study area. This is done through the built-in variable **shape** of type geometry.

There is only ONE, global geometry that is called **shape**. Usually, you want to define it as the bounding box (GAMA: envelope) of your GIS data.

```
//define the bounding box
geometry shape <- envelope(pasture_file);
```

Set the spatial and temporal scale

```
//the temporal granularity is specified by means of the duration of 1 time step
float step <- 1 #minute;

//set meters as the unit of the cow‘s action radius
float action_radius <- 10.0#m;
```

Create geometries from vector data

In the global section declare a geometry variable

```
    geometry pasture_polygon;
```

And in the global init {} assign the polyon shape:

```
    pasture_polygon <- geometry(pasture_file);
```

Restrict cows to their pasture

In the cows species section, you can use the habitat geometry to have the movement of lynx bound to the habitat:

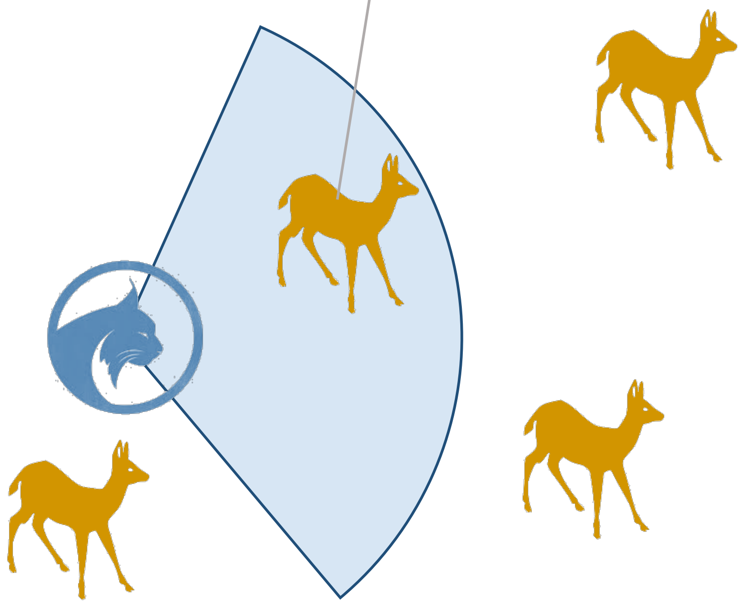
```
    do wander amplitude: 60.0 speed: cow_speed bounds: fenced_area;
```

Agent-agent interaction

Agents interact, once they are close.

For example:

IF in the vision of a lynx is a deer -> hunt it.



The predator‘s perceived area **overlaps** the prey‘s location.

Agent-agent interaction

Needs to call one agentset from another agentset

Variables of (lists of) agents & spatial operators do the trick!

```
//agents definition of lynx, deer and forest
species lynx skills:[moving]{
  geometry perception_area;
  // the deer that the lynx will hunt
  deer my_deer;
  //behaviour of the lynx: walking around in correlated random walk or hunt if deer is in
perception_area
  reflex behaviour{
    perception_area <- (self + range) intersection cone (int(heading - 60), int(heading + 60));
    my_deer <- (deer overlapping perception_area) closest_to self;
    if (my_deer != nil){
      do hunt;
    }
}
```

each lynx “owns” as specific deer, or a list of deers

buffer geometry

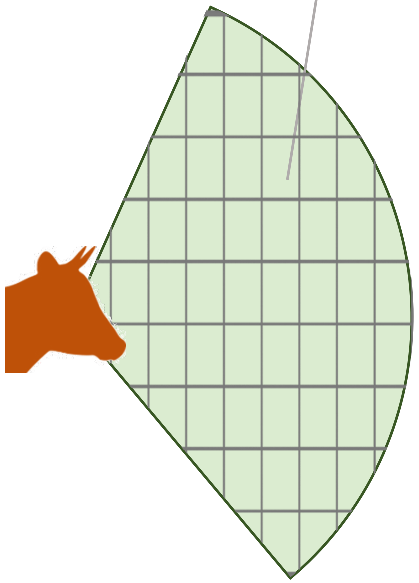
Spatial operators

Agent-environment interaction

Agents interact with their local environment.

For example:

IF in a cow‘s action-radius is grass -> graze



The cow‘s perceived area **overlaps** grass cell locations.

Raster environment: **grid** vs. **field**

A **grid** is a species of type raster, where each of the cells is an „agent“ with built-in attributes and operators.

Powerful, but highly memory consuming and computationally expensive

```
grid grass cell_width:50 cell_height:50 neighbors:6 {..}
```

A **field** is a variable of type field that can be accessed by their location.

A „lightweight“ possibility to represent the environment

Well suited for geographic raster data

New in GAMA; the documentation is lagging behind..

To construct a field 50 by 50 cells with a default value of 0:

```
field biomass <- field(50,50,0.0);
```

Looping through grids/fields

To loop through the cells of a grid you can use the ask statement:

```
ask grass {}
```

To loop through the cells of a field, you need to use a loop:

```
loop s over: biomass{
  biomass[centroid(s)] <- 6;
}
```

Loop through a subset of cells of grids/fields

To loop through the cells of a grid you can use the ask statement:

```
list<grass> pasture_cells <- grass overlapping pasture_geom;
ask pasture_cells {
  biomass <- 6; //biomass is a grid variable in this example
}
```

To loop through the cells of a field, you need to use a loop:

```
loop s over: biomass cells_in pasture_geom{
  biomass[centroid(s)] <- 6;
}
```

Agent interaction with ONE cell

Needs to call one agentset from another agentset

Lists & spatial operators do the trick!

```
species cows skills: [moving] {
    point best_spot;
    // cow interaction with grass: each cow has its “best spot”
    reflex graze{
        best_spot <- ((biomass cells_in (self + range) collect (each.centroid))) with_max_of each.z;
        …
    }
}
```

A variable of type grass is declared as an attribute of a cow

field

buffer around cow

z holds the attribute of the field cell’s centroid

spatial operator

evaluate each element in the list

Agent interaction with ONE cell

Needs to call one agentset from another agentset

Lists & spatial operators do the trick!

```
species cows skills: [moving] {
    point best_spot;
    // cow interaction with grass: each cow has its “best spot”
    reflex graze{
        best_spot <- shuffle(((biomass cells_in (self + range)) collect (each.centroid))) with_max_of each.z;
        …
    }
}
```

A variable of type grass is declared as an attribute of a cow

avoid direction bias

buffer around cow

z holds the attribute of the field cell’s centroid

Evaluate each element in the list

Spatial operator

Agent interaction

```
species cows skills: [moving] {
  float energy;
  point best_spot;
  //cows graze 2 units of biomass per time step, if possible
  reflex graze {
    //identify the field cell with the maximum biomass within the range of the cow
    best_spot <- shuffle(((biomass cells_in (self + range)) collect
(each.centroid))) with_max_of each.z;
    do move heading: self towards best_spot;
    biomass[self.location] <- biomass[self.location] - 2;
  }
}
```