Ecosystem Simulation

MEIC - 2021/22 - Agents and Multi-Agent Systems - 2nd Assignment

João Carlos Machado Rocha Pires (UP201806079)
Rafael Valente Cristino (UP201806680)

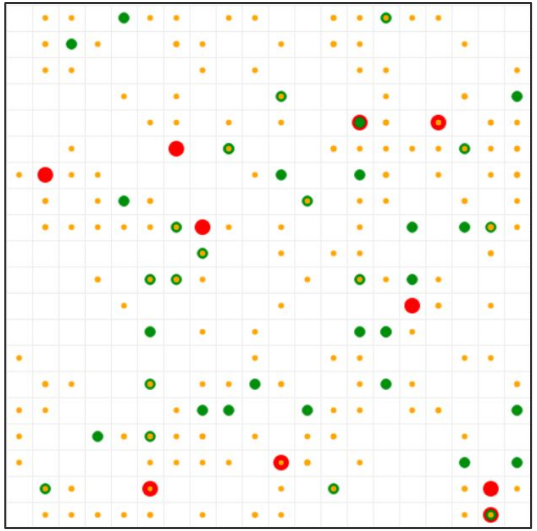
U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Problem description

Classic predator-prey simulation. Our system contains three types of agents: carrots (**vegetation**), rabbits (**prey**) and foxes (**predator**). The vegetation spawns randomly. Both the prey and the predators seek to stay alive and reproduce. The predators eat the preys and the preys eat the vegetation.

The objective is to vary multiple parameters (e.g. the initial number of specimens of each species, their characteristics and their behaviours) and observe the emergent properties of the system.

Agents have energy and age. If they become too old they die. They spend energy on each move and get energy when they eat. If the energy reaches 0, they die.



Setup

On startup, the environment is populated with the specified amounts of agents of each species, each of them in a random position.

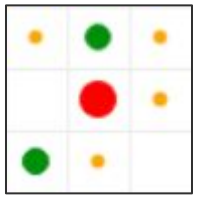
The amount of vegetation is kept constant throughout the execution of the simulation by populating it on each step.

Problem description

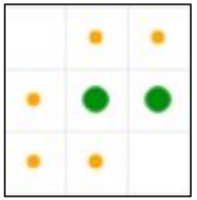
Decision making

On each step, each active agent (predators and preys) needs to make a move, taking into account their respective neighbourhood:

- If the agent needs to eat (his energy is less than the specified threshold) and the neighbourhood contains food, he moves to that spot and eats.
- If the agent doesn't need to eat, is ready to mate (the time since the last mating is greater than the specified mating timeout) and the neighbourhood contains an agent of the same species who is also ready to mate, he moves to that spot and mates.
- If the agent doesn't need to eat and isn't ready to mate, he moves to a random adjacent cell.



Neighbourhood example of a predator (in the center)



Neighbourhood example of a prey (in the center)

Independent Variables

```
EcosystemParameters(
    predator_parameters=AgentParameters(
        initial_energy_amount=10,         # the initial energy of the agent

        #predators aren't eaten
        energy_given_when_eaten=0,        # how much energy agents that eat it receive

        max_age=60,                       # if older than max_age (in steps), the agent dies

        mating_timeout=10,                # minimum timeout (in steps) between each mating
        min_energy_to_mate=8,             # minimum energy that the agent needs in order to mate

        energy_threshold_to_eat=10        # if the agent's energy is larger than this value he won't actively seek food
    ),

    prey_parameters=AgentParameters(...),

    vegetation_parameters=PassiveAgentParameters(
        initial_energy_amount=10,
        energy_given_when_eaten=10
    ),

    grid_width=20,
    grid_height=20
),
initial_predator_amount=5,
initial_prey_amount=40
vegetation_amount=200
```

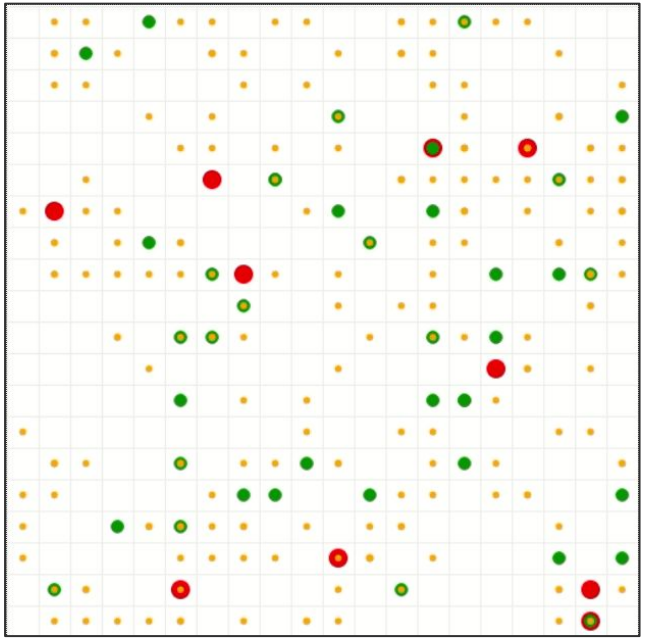
Dependent Variables

As dependent variables we have the emergent properties of the system. These properties are measured and plotted in real-time during the execution of the simulation.

- The most important property is the **amount of prey and predators**. Optimally we would be able to achieve an equilibrium that allows both prey and predators to coexist.
- The **mean age of prey and of predators** throughout the execution.
- The **mean amount of energy of prey and of predators** throughout the execution. This is useful to understand the flow of energy throughout the system.

Interaction space and simulation execution visualization

The interaction space is a **toroidal grid** (edges wrap around) and each grid space may contain more than one agent - we used the **`mesa.space.MultiGrid`**. For the visualization we used the **mesa visualization server** and its real-time plot functionalities.

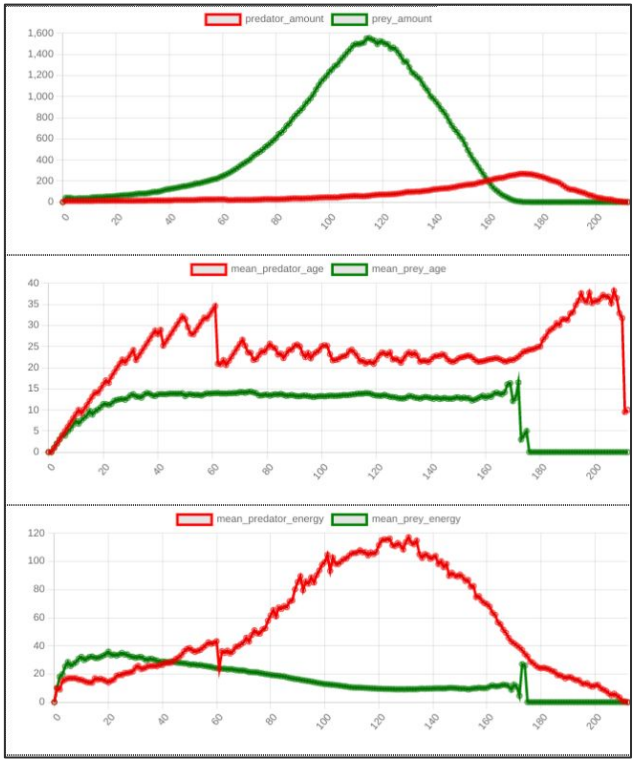


Real-time grid visualization (Predator Prey Vegetation)

predator and prey amounts

mean predator and prey ages

mean predator and prey energy amount



Real-time plot visualization

Experiences

Using the **batch_run** function that MESA provides, we ran the model with 8 different environment configurations.

For all those configurations, the following variables were kept unchanged:

Predator Parameters	Prey Parameters
Initial energy amount: 10 | Energy given when eaten: 0 Max age: 60 | Mating timeout: 10 | Min energy to mate: 8 Energy threshold to eat: 10	Initial energy amount: 10 | Energy given when eaten: 10 Max age: 75 | Mating timeout: 5 | Min energy to mate: 5 Energy threshold to eat: 20
Vegetation Parameters	**Grid Parameters**
Initial energy amount: 10 | Energy given when eaten: 10	Grid width: 20 | Grid height: 20

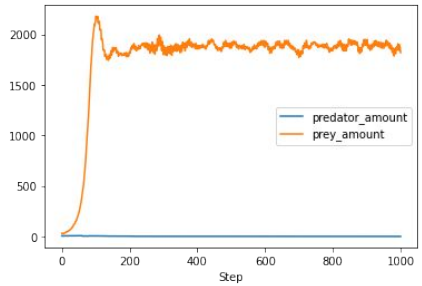
The configurations were different in the number of initial **preys**, **predators** and **vegetation** (next slide).

For each of those configurations, we ran one iteration per configuration with the max. number of steps being 1000 and collecting the data after each step of each configuration.

The max. number of steps means that if the ecosystem simulation does not end before the 1000th step, it will end then. To end, the simulation must reach a point where the number of **preys** and **predators** is simultaneously equal to 0.

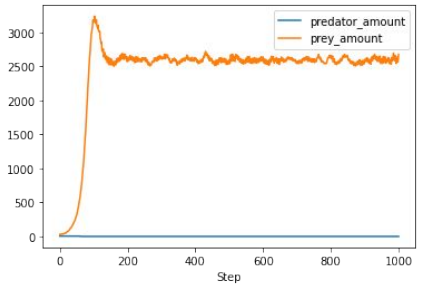
Results

Initial amounts of each agent: **Predator** **Prey** **Vegetation**



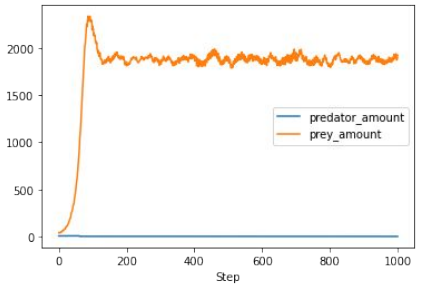
Configuration 1

5 **30** **150**



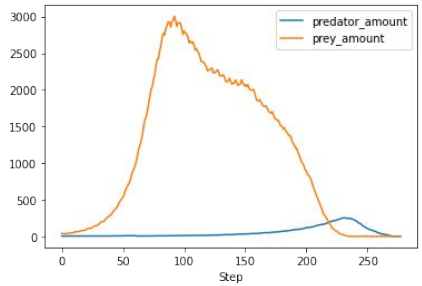
Configuration 2

5 **30** **200**



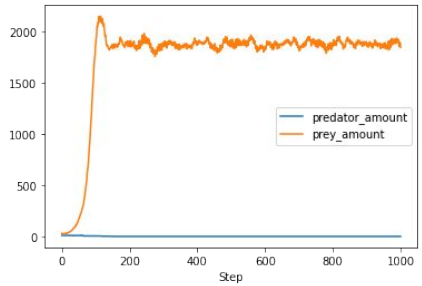
Configuration 3

5 **40** **150**



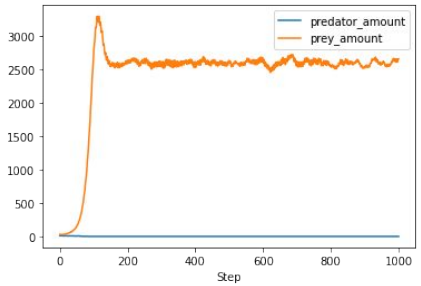
Configuration 4

5 **40** **200**



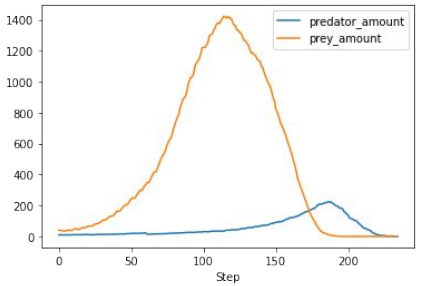
Configuration 5

10 **30** **150**



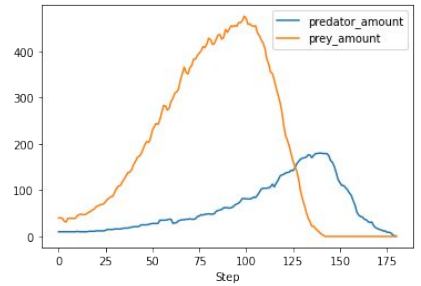
Configuration 6

10 **30** **200**



Configuration 7

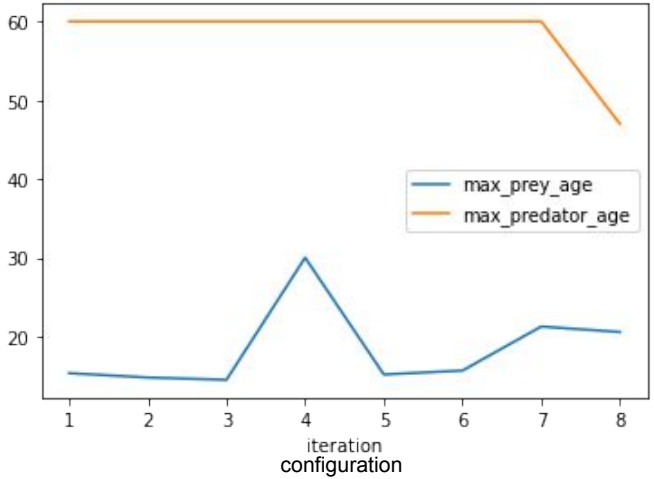
10 **40** **150**

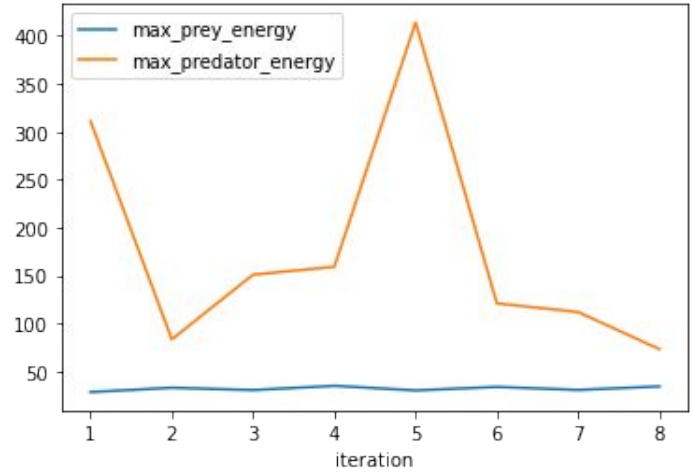


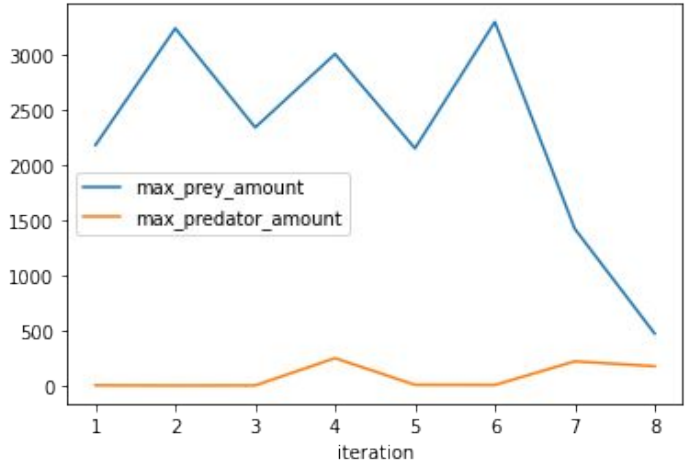
Configuration 8

10 **40** **200**

Results

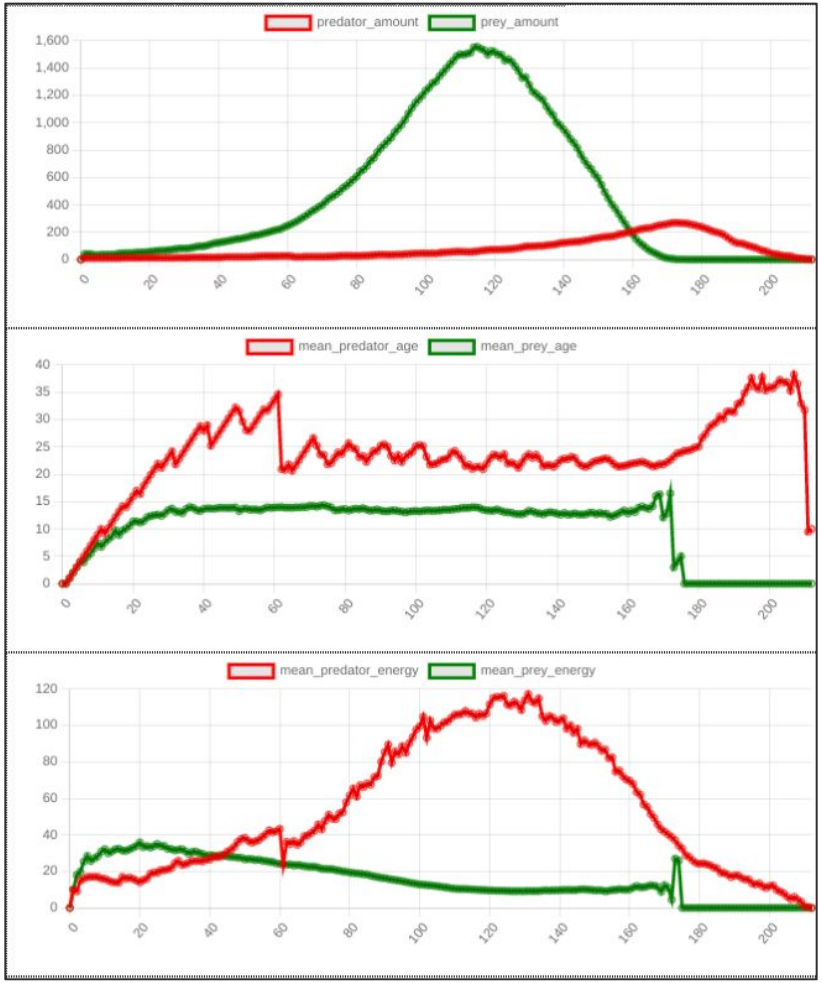






Configuration	Steps
1	1000
2	1000
3	1000
4	277
5	1000
6	1000
7	234
8	180

(extensive analysis in the submitted Jupyter Notebook)



Real-time plots obtained using configuration 7 (**10** **40** **150**)

Results

Emergent System Properties

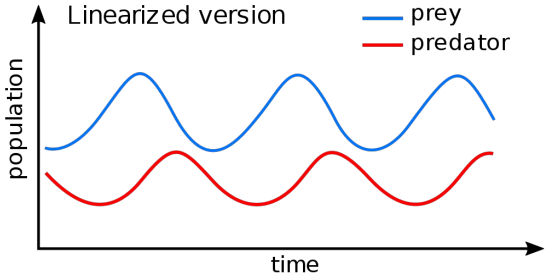
- An increase in the amount of preys is followed by an increase in the amount of predators. An increase in the amount of predators results in a decrease in the amount of preys. In an equilibrium this would originate an oscillating plot resembling a trigonometric function (depicted in the next slide).
- While the agents are reproducing, the mean age is lower. When the preys are scarce, the predators mean age increases (less reproduction). When the preys are all being eaten, their mean age reduces.
- The mean energy of the predators resembles the curve of the amount of preys - the more preys there are, the more they eat and the more energy they get.

Conclusions

- Reaching a state of equilibrium is harder than it looks.
- Small changes in any of the parameters provoke great change in the outcome of the simulation.
- The preys seem to be more stable, i.e. the amount of preys, their energy and age values remain more stable over all the executions with different parameters.
- In all the iterations whose maximum number of steps was lower than the maximum defined by us (1000), the graphic of the amount of predators and preys resembles the one below.

Future work

- Achieving a population equilibrium, without extinction.
- Use of reinforcement learning for deciding each agent's next movement (change to strategic position or remain in the same).
- Execute more simulations with more variety in environment configurations.



Population equilibrium plot reference:
https://en.wikipedia.org/wiki/Lotka%E2%80%93Volterra_equations

Additional Information

Detailed execution examples

Execution examples here.

- Configuration 1 (here)
  - Initial agent amounts: 5 predators, 30 prey, 150 vegetation
- Configuration 7 (here)
  - Initial agent amounts: 10 predators, 40 prey, 150 vegetation

How to run

- To run the simulation with visualization, one can use the “`mesa runserver .`” command in the root code folder.
- The simulation can also be ran in batch, as done in the provided jupyter notebook.
- One can also run the simulation through code by importing and instantiating the mesa model that was developed:

```
from model import EcosystemModel


model = EcosystemModel(parameters)

for _ in range(number_of_steps):

    model.step()
```

Code Excerpts

Function used to move the agents (agents.BeingAgent:ActiveBeingAgent)

```
def move(self, to_mate: AgentType, to_eat: AgentType):
    possible_steps = self.model.grid.get_neighborhood(
        self.pos,
        moore=True, # moore includes all 8 surrounding squares, von neumann only top right left bot
        include_center=True # the agent can remain stationary
    )

    agents_in_neighbourhood = self.get_agents_in_neighbourhood(possible_steps)

    if len(agents_in_neighbourhood[to_eat]) > 0 and self.needs_to_eat():
        new_position = self.random.choice(agents_in_neighbourhood[to_eat])
    elif len(agents_in_neighbourhood[to_mate]) > 0 and self.can_mate():
        new_position = self.random.choice(agents_in_neighbourhood[to_mate])
    else:
        new_position = self.random.choice(possible_steps)

    self.model.grid.move_agent(self, new_position)
    self.decrement_energy()
```

Code Excerpts

Prey step function (agents.PreyAgent:PreyAgent)

```
def step(self):
    self.increment_attributes() # increment age and time since mating
    if self.is_dead(): return

    self.move(AgentType.Prey, AgentType.Vegetation) # move(same species, food)
    cellmates = self.get_agents_in_same_cell()

    if len(cellmates[AgentType.Predator]) != 0: return

    self.eat_first_alive_agent(cellmates[AgentType.Vegetation])
    self.mate_with_first_free_agent(cellmates[AgentType.Prey])
```

Code Excerpts

Predator step function (agents.PredatorAgent:PredatorAgent)

```
def step(self):
    self.increment_attributes() # increment age and time since mating
    if self.is_dead(): return

    self.move(AgentType.Predator, AgentType.Prey) # move(same species, food)
    cellmates = self.get_agents_in_same_cell()

    self.eat_first_alive_agent(cellmates[AgentType.Prey])
    self.mate_with_first_free_agent(cellmates[AgentType.Predator])
```