DS 3500: Advanced Programming with Data
Prof. Rachlin
Northeastern University

Ecosystem Modeling

Leveraging the power of animation and simulation to explore complex population dynamics


Introduction

In class we created a very simple model of rabbits hopping around a field of grass. Rabbits eat the grass. Overtime, the grass grows back. Rabbits who eat survive and reproduce. Rabbits who do not eat starve and die off. Even this simplified model ecosystem, with its basic rules for moving, eating, surviving, and reproducing was sufficient to demonstrate that the population dynamics and the spatial distribution of the rabbits is highly complex. In this homework, you will extend the model, step-by-step, to support more realistic model ecosystems involving both predators (foxes) and prey (rabbits). What happens when foxes are introduced to prey upon the rabbits? Implement the updated model, run your simulations, and report your results.

Model Changes

1. Change the "Rabbit" object to Animal. We will re-use the Animal object to represent both foxes and rabbits. (Alternatively, we could define a base class Animal and have Fox and Rabbit objects that extend the Animal base class, but I think this will be easier.) The Animal constructor includes parameters for max_offspring (default=1), starvation_level (default=1), and reproduction_level (default=1). These three parameters are saved as state variables in the Animal object. Set a fourth state variable: self.hunger = 0. Set a fifth state variable: self.alive = True.

2. When an animal goes a generation without eating, its hunger level increases by one. If the hunger level reaches the starvation level, the animal dies. In the starting model, note that rabbits die immediately if they do not eat (starvation_level=1). On the other hand, if the animal eats something, the hunger level is reset to zero.

3. Animals can only reproduce if the amount they have eaten is at least as high as the reproduction level. When animals reproduce, their "eaten" level is reset to zero. Reproduction requires a lot of energy, apparently!

4. Create a custom color map to display the state of the field and the location of the rabbits and foxes:

    0: Black (Nothing at that location)
    1: Green (Grass but no animals)
    2: White (Rabbit – but no foxes)
    3: Red (Fox)

    When rendering the field, display the locations of rabbits and foxes. If rabbits and foxes are at the same location, display it as a red point (fox). Note: remember that the field state object (Field.field) should NOT be changed with animal locations. It is always an array of zeros and ones to represent the location of the grass only. Instead, make a copy and overlay rabbit (2) and fox (3) locations. Be sure to update vmin and vmax in your call to plt.imshow!

5. When a fox lands on the same location of a rabbit, that rabbit is now dead. Mark it dead and remove it from the population as part of determining which animals survived to the next generation. When feeding foxes, likely you will need a data structure mapping (x, y) location to rabbits at that location. If you otherwise try to scan through the list of rabbits looking for a location match, your code will run very slowly! If two foxes are at the same location where there are rabbits, assume that they both consume all the rabbits at that location.

What to Submit

1. Your code: **alife.py**

2. Your model assumptions including, for both rabbits and foxes:
    a. Initial numbers
    b. Max number of offspring
    c. Starvation level (# generations the animal can go without eating)
    d. Reproduction level (Amount of food the animal must eat to reproduce)

    Note: You will need to tinker with these assumptions to find a stable model.

3. Your written analysis: Write a 1-2-page report documenting the assumptions underlying your stable fox/rabbit model. Show how the landscape looks at various generational intervals. Also create a time series plot showing how the number of foxes and rabbits changes overtime. Interpret this plot as part of your analysis.