Foxes and Rabbits

Overview

Exam 2021 - Group Project

DM536 Introduction to Programming
DM562 Scientific Programming
DM857 Introduction to Programming
DS830 Introduction to Programming

A predator-prey model

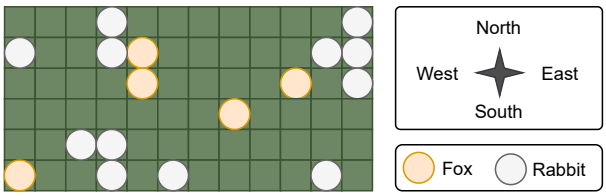
Predator-pray models describe the evolution over time of the populations of two species of which one (the preys) is the main source of food of the other (the predators). These models have applications to many areas such as chemistry, ecology, biomathematics, economy, optimisation. There is a vast corpus about such models and their applications and many approaches to studying their properties chiefly via stochastic simulations (simulations with variables that vary randomly) and via differential equations (e.g., Lotka-Volterra equations).

This project will focus on simulations and make the following simplifying assumptions about the model:

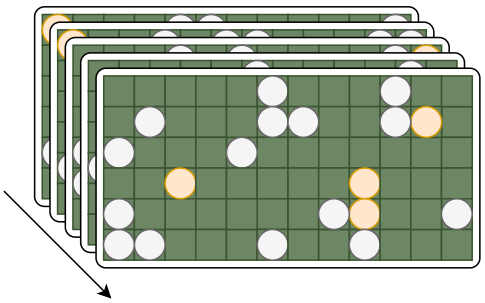
1. The food supply for the prey population is replenished at a constant rate.
2. The food supply for the predator population depends on the prey population.
3. Reproduction is asexual: a mature individual reproduces with a fixed probability provided that it is well-fed and there are no predators nearby.
4. The environment does not change over time in favour of either population making evolution inconsequential.

Foxes and Rabbits

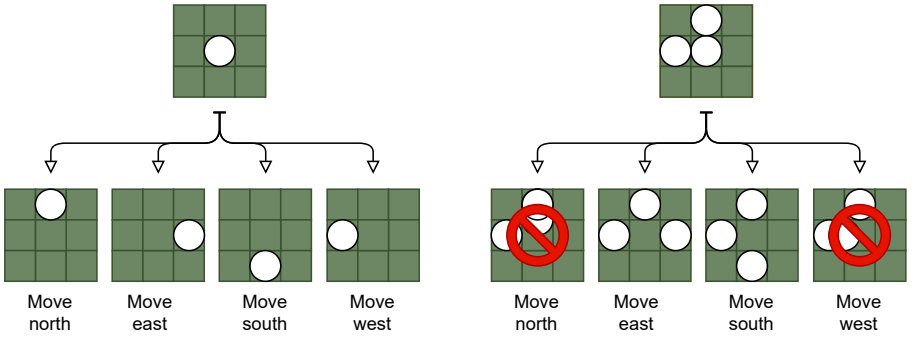
The environment in this simulation is a finite 2D grid (a rectangle or a toroid) with a fixed size. The coordinate system is oriented north-to-south and west-to-east. On this grid we find foxes (predators), rabbits (preys) and grass. On each cell there can be at most one fox and at most one rabbit (a fox and rabbit can be on the same spot at the same time but not two rabbits or two foxes). There is a fixed limit on the amount of grass on each cell.



This grid evolves over time in discrete steps: at each tick of the simulation clock animals move, feed, reproduce, and grass grows.



Animals can move along the north-south and west-east directions one cell at a time, provided that the destination cell is not occupied by an animal of the same species. Below are all the possible evolutions of a 3x3 grid where the animal in central cell takes a step. On the left all four directions are free whereas on the right the north and west directions are blocked by the presence of an animal of the same species.

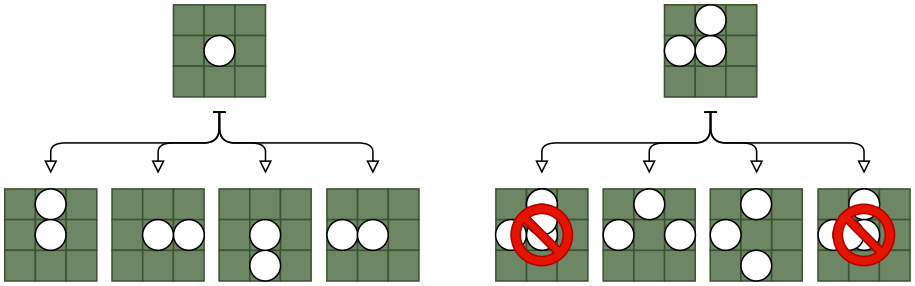


The choice of destination is random following a uniform probability distribution.

Animals can reproduce at any time provided that:

1. they reached maturity (they have at least a certain age),
2. they are well-fed (they have eaten enough food),
3. they are safe from predation (a fox is always safe, a rabbit is safe if there are no foxes in its cell and the ones adjacent to it),
4. one of the adjacent cells can host the newborn.

The cell for the newborn is selected randomly.

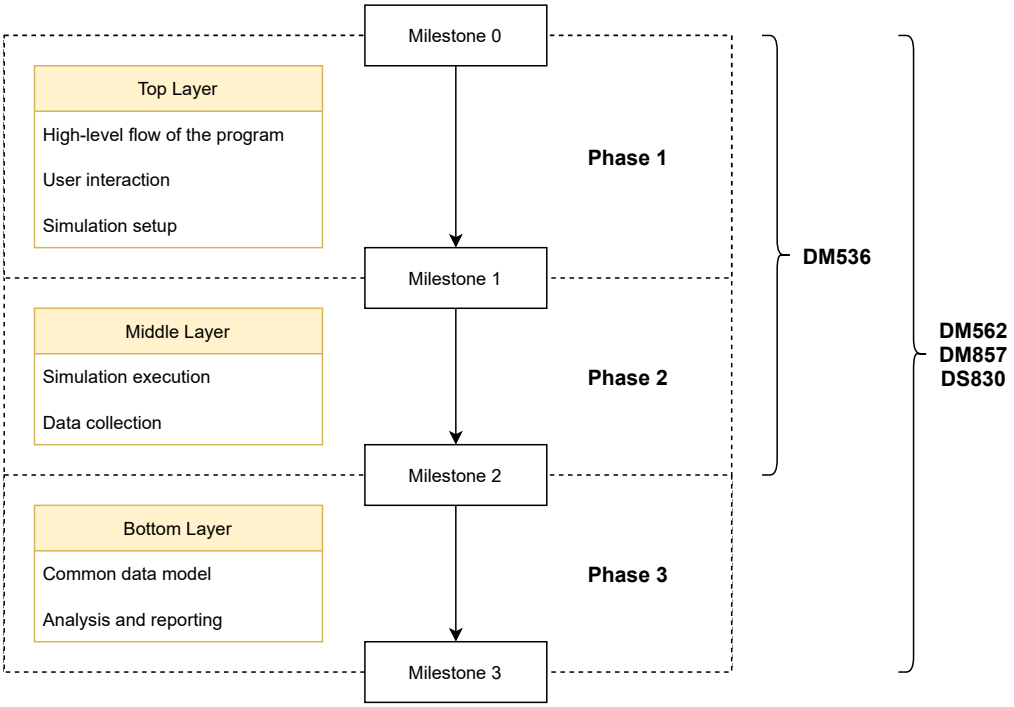


Both foxes and rabbits have a level of energy that is replenished eating food. A fixed amount of energy consumed at each tick of the simulation clock (animal metabolism) and during reproduction. If an animal runs out of energy it dies of starvation. The maximum amount of energy an animal can have is fixed.

All animals have an age and the maximum age for a foxes and for rabbits is fixed. When an animal reaches the maximum age for its species it dies of old age.

Project workflow

For this project you will follow a top-down development process where the program implementation is refined as the development moves from the top to the bottom layer. Each layer is the object of a separate phase of the project workflow.



Each phase ends with a milestone (1,2, and 3) and produce a version of the program where functionalities of the corresponding layer (and the ones above it) are fully implemented and working when supplied with an implementation (complete or mock[1]) of the lower layers. For instance, during Phase 1 you will implement the functionality of the top layer and use some place-holder version of the middle and lower layer. If someone replaces the mock versions of the middle and lower layer with full implementations, then the resulting program must be a full implementation.

At the beginning of each phase, you will receive detailed instruction covering

- the specification of the components and functionalities you need to implement,
- which components you need to mock,
- the expected output for the milestone.

[1] A mock implementation of a component is a place-holder version of the component that only checks the validity of inputs and produces legal but arbitrary outputs for the sake of testing.