Programming Project

Due July 11, 2020, Midnight(23:55)

Requirements

- Implementation
- Documentation
- Video Presentation

The goal for this programming project is to create a simple 2D predator-prey simulation. In this simulation the prey are ants and the predators are doodlebugs. These critters live in a world composed of a grid of cells(grid size should be adjustable by changing constant global variables). Only one critter may occupy a cell at a time. The grid is enclosed, so a critter is not allowed to move off the edges of the world. Time is simulated in time steps. Each critter performs some action every time step.

The ants behave according to the following model:

- Move. Every time step, randomly try to move up, down, left or right. If the neighboring cell in the selected direction is occupied or would move the ant off the grid, then the ant stays in the current cell.
- Breed. If an ant survives for three time steps, then at the end of the time step (i.e. after moving) the ant will breed. This is simulated by creating a new ant in an adjacent (up, down, left, or right) cell that is empty. If there is no empty cell available then no breeding occurs. Once an offspring is produced an ant cannot produce an offspring until three more time steps have elapsed. There is a possibility of mutation. Randomly, with a small probability, the new ant can be a mutated version. Randomly mutate an ant make it poisonous. If a poisonous ant breeds, the new ant is also poisonous.

The poisonous ants behave according to the following model:

- Move. Every time step, randomly try to move up, down, left or right. If the selected direction would move the ant off the grid, then the poisonous ant stays in the current cell.
- Breed. If a poisonous ant survives for four time steps, then at the end of the time step (i.e. after moving) the ant will breed. This is simulated by creating a new poisonous ant in an adjacent (up, down, left, or right) cell that is empty. If the randomly selected cell is not empty, the poisonous ant looks for other adjacent cells. If the selected cell is empty a new poisonous ant is created in the selected cell. If all of the adjacent cells are occupied, the new poisonous ant fights and kills the occupant in the last cell tried(killing means replacing) Once an offspring(the new poisonous ant) is produced an ant cannot produce an offspring until four more time steps have elapsed.

The doodlebugs behave according to the following model:

- Move. Every time step, if there is an adjacent ant (up, down, left, or right) then the doodlebug will move to that cell and eat the ant. Otherwise the doodlebug moves according to the same rules as the ant. Note that a doodlebug cannot eat other doodlebugs.
- Breed. If a doodlebug survives for eight time steps, then at the end of the time step it will spawn off a new doodlebug in the same manner as the ant.
- Starve. If a doodlebug has not eaten an ant within the last three time steps, then at the end of the third time step it will starve and die. The doodlebug should then be removed from the grid of cells.
- A doodlebug can eat ants and poisonous ants. If a doodlebug eats a poisonous ant, it can only live two time steps.

During one turn, all the doodlebugs should move before the ants. All the ants move before the poisonous ants. Write a program to implement this simulation and draw the world using ASCII characters of “o” for an ant, “X” for a doodlebug and “c” for poisonous ant. Create a class named Organism that encapsulates basic data common to both ants, and doodlebugs. This class should have a virtual function named move that is defined in the derived classes of Ant and Doodlebug. Derive poisonous ant class from the ant class. You may need additional data structures to keep track of which critters have moved.

Initialize the world with 5 doodlebugs and 100 ants. No poisonous ants initially. After each time step prompt the user to press enter to move to the next time step. You should see a cyclical pattern between the population of predators and prey, although random perturbations may lead to the elimination of one or all species.

- Apply as much as object oriented design principles as possible.

- Implementation should be distributed to multiple files.
- Documentation is important. Create a pdf file which describes your design and implementation details.
- Show the execution of your program by video capturing your screen. You can present it by recording your audio too. If you don't want to record your audio, add on-screen explanations to the video. Upload the video to a video hosting platform.
- Submit your implementation, documentation and video link info in a zip file.