

CS162 Lab 5

Goals

- Refine requirements for a program using inheritance
- Implement the requirements using polymorphism and OOP

In this lab, you will use cellular automata to create a **2D predator–prey simulation** in your program. The preys are ants and the predators are doodlebugs. These critters live in a 20 * 20 grid of cells. Only one critter may occupy a cell at a time. The grid is enclosed and no critter may move off the grid. Time is simulated in steps. Each critter performs some action every time step.

The ants behave according to the following model:

- **Move:** For every time step, the ants **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 (not been eaten by doodlebugs), 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 randomly**. If there is no empty cell available, no breeding occurs. Once an offspring is produced, an ant cannot produce an offspring again until it has survived three more time steps.

The doodlebugs behave according to the following model:

- **Move:** For every time step, the doodlebug will firstly try to **move to an adjacent cell containing an ant and eat the ant** (you can decide if there are several ants in the adjacent cells, how the doodlebug will choose to move). If there are no ants in adjacent cells, 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, 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 three time steps**, at the end of the third time step it **will starve and die**. The doodlebug should then be **removed** from the grid of cells.

Initialize the world with **5 doodlebugs and 100 ants**. You will randomly place them. You should see a cyclical pattern between the population of predators and prey, although random perturbations may lead to the elimination of one or both species. You will **prompt the user to enter the number of time steps to run**.

Create a class named **Critter** that contains data and functions **common to ants and doodlebugs**. This class should have a **virtual function** named **move** that is defined in the **derived classes of Ant and Doodlebug**. Each class will be in its own source file.

For each time step, do the following in your program: after moves, when breeding, eating, and starving are resolved, **display the resulting grid**. Draw the world using ASCII characters of **“O” for an ant, “X” for a doodlebug and “E” for an empty space** (the characters should be arranged to look like a grid). The doodlebugs will **move before** the ants in each time step. When you reach the time steps entered by the user, ask them to enter another number and start to run the simulation again or to exit the program. You must maintain the state of the current grid while creating the next display. You will use a **dynamic array** to represent the grid. Each array element will be a pointer to a Critter. Get your program running and tested. For debugging your program, you should save the random placement until you have everything else working properly. In general, “random” is bad for testing and debugging.

Extra Credit: In addition to prompting the user for the number of time steps, ask them to enter the **size of the grid rows and columns, the number of ants, and the number of doodlebugs**. If you did the extra credit part, **you must print a line on the screen at the beginning of your program to inform the grader that you did it**.

For this lab, you do not need to write a report. Turn in all your program files and makefile (add a README file if necessary) in a zip file on TEACH.

Grading

- Programming style - 1 point
- Ant class is created and works correctly - 2 points
- Doodlebug class is created and works correctly - 2 points
- Virtual move() function works correctly - 3 points
- Simulation runs correctly, displaying the changes for each time step - 2 points
- Extra credit: Allow the user to set the grid size, # of ants, and # of Doodlebugs - 1 point