Group 13 (Joel Huffman, Andrew Swaim, Frank Li, Henry Clay)
CS162 Intro to CS II

2D Predator-Prey Simulation

Design and Plan

Menu Class or function

- Prompts the user and gets information from them including:
    - Number of steps to run simulation for
    - Play another game or quit
    - If we do extra credit:
        - Get grid size
        - Get # of ants to spawn
        - Get # of Doodlebugs to spawn
        - Print extra credit message at beginning of program

Input Validation Class or function

- Checks if user input is valid type and within acceptable bounds
- Prevents program from crashing
- Reprompts user until valid input is received.

Critter Class

- Parent/base class of Ant and Doodlebug
- Class Variables
    - positionX
    - positionY
    - daysSinceLastBred
- Member Functions
    - Move() (needs to be virtual)
    - Breed() (needs to be virtual)
    - randomDirection(): generates a random direction (int 0-3?) and returns that variable. Can be used for breeding and moving.
    - checkSpace(int x, int y, char lookingFor): checks to see if space is containing lookingFor). Can be used for moving, eating and breeding.

Ant class

- Inherits from Critter class
- Class variables
    - daysSurvived
- Member Functions

- Move: For every time step, the ants **randomly** 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 the cell that is picked is not empty, randomly attempt to pick another one. 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

Doodlebug Class

- Class variables
  - daysSurvived
  - daysSinceEating
- Member Functions
  - 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 (the Doodlebug will only breed into an empty cell).
  - 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.

Game Class

- If doing extra credit:
  - Create the gameboard from user entered values (height, width, # of doodlebugs, # of ants)
- Creates the 2D gameboard from a dynamic array of pointers to Critter objects
- Places ant and doodlebugs randomly
- Class variables
  - boardWidth
  - boardHeight
- Member Functions
  - randomPlace(Critter* c): chooses a random space to place a doodlebug or ant. If the space is already occupied, another random spot is chosen until a free space is found.

- drawBoard(): draws the gameboard in its current step of the simulation.

Main

- Bring the whole project together and make sure the everything runs smoothly

Write Reflection

- Include design document and outline
- Write, run and record test cases
- What problems we ran into
- How were problems fixed
- What did we learn

Final touches

- Create makefile
- Ensure program works on FLIP
- Check for memory leaks
- Zip files
- Submit project

Pseudo-code

```
Greet user
Display intro
Display instructions/purpose
Display indication of extra credit for the grader

Do {
        Prompt for # of rows
        Prompt for # of cols
        Prompt for # of DBugs
        Prompt for # of Ants
        Validate that the # of bugs + ants < rows * cols (to make sure there are more spaces
than critters)

} while inputs are invalid

// Start the program - loop back to here if the user wants to run again
Do {

        Prompt for # of steps

        Create the grid
        Create and randomly place the critters
        // Start the simulation for the # of steps entered by the user
        Do {

                //DBugs move first.
                Move Dbugs {
                        Check surrounding spaces for random ants
                                If found move to space of ant and remove (eat) ant
                                        Reset daysSinceEating to 0
                        Check if DBugs starved {
                                Check if DBug's daysSinceEating is >= 3
                                        If so, then delete this object
                        }
                        If no ant found, pick a random direction
                                Increment daysSinceEating by 1
                                Check if the space in the selected direction is empty
                                AND that the DBug is not against the wall
                                        If so move the DBug to the empty space.

                }
```

```
        Move Ants {
            Pick a random direction
                Check if the space in the selected direction is empty
                AND that the Ant is not against the wall
                    If so move the Ant to the empty space.
        }

        Decrease step time by 1

        Breed DBugs {
            Check if days survived is >= 8
            Find random space around DBug that is empty
                If at least one empty space, then make new DBug()
                    Reset days (time step) survived to 0
                If no open space, then don't do anything
        }
        Breed Ants    {
            Check if days survived is >= 3
            Find random open space around Ant that is empty
                If at least one empty space, then make new Ant()
                    Reset days (time step) survived to 0
                If no open space, then don't do anything
        }

        Display grid

        If there are remaining steps
            Hang and prompt user to continue or quit early

    } while # of steps > 0

    Deallocate memory for grid and critters

    Prompt if the user would like to enter a new # of steps and run simulation again or quit
the program.

} while userchoice != quit

Display the outro
Exit program
```

Test Table

Test Case	Input Values	Functions involved	Expected outcomes	Observed outcomes
Board is completely populated with ants	5 5 0 25 10	runSim()	Simulation terminates with early termination message	Simulation terminates with early termination message
Board is completely empty	5 5 0 0 10	runSim()	Simulation terminates with early termination message	Simulation terminates with early termination message
Attempt to place more ants than available spaces	5 5 15 30	isInputInt(int ceiling, int base)	Print “Out of range - enter integer between 0 and 10. Try Again!”	Print “Out of range - enter integer between 0 and 10. Try Again!”
Entering non-integer values as input	F % a	isInputInt(int ceiling, int base)	Input not accepted, try again!	Input not accepted, try again!
Doodlebugs can consume Ants if available	10 10 5 25 50	runSim()	Ants are deleted and Doodlebugs move into position	Ants are deleted and Doodlebugs move into position
Ants & Dbugs are able move around and breed when supposed to	10 15 10 10 50	Virtual move() runSim() breed()	Critters can traverse board without memory issues and breed at appropriate times	Critters can traverse without memory issues and breed at appropriate times
Critters are randomly placed at beginning of program	20 20 5 10 25	randomlyPlace()	With same arguments each time, the Critters are placed differently	With same arguments each time, the Critters are placed differently and do not “repeat”

Check for memory leaks (valgrind)	20 20 5 10 2000	~Board() destructor	Any kind of termination results in all memory blocks being freed	Any kind of termination results in all memory blocks being freed

Reflection

Instructions on what a reflection should address from Project 1 assignment:

1. You should explain what design changes you made
2. How you solved some problems you encountered
3. What you learned for this project

For our group project, we first began our planning phase within Canvas. The features available on Canvas included a discussion thread and a place to upload files that could be organized into unique folders. Aside from that, the tools provided there were quite limiting. We used the thread for quick introductions and moved to the design phase within Google Docs and Slack for conversing about the project.

During the design phase, we jotted down the general requirements and possible options for implementation for the various classes that would need to eventually be coded. The natural next step was to start writing pseudocode to get our logic on paper. In retrospect, the pseudocode seemed comprehensive at the time but, after reaching the end of the project, with working code, the designs evolved as everyone's changes compounded over time. In regards to our pseudocode, many of the changes were with the order of flow when executing the program. For example, we initially placed the reset of days going without eating next to the time step incrementation line, but it later made more sense to include the reset right when the Doodlebug deleted an Ant and took over the new space. While couple the timestep with the reset together could somehow work in the end, we made these pseudocode changes so that our methods would eventually be as consolidated as possible with regard to related functionality.

One of the biggest problems we as a team needed to resolve was that, by the near end of the project, we were running into trouble getting our Doodlebugs to move and move correctly. Each member on the team attempted to debug this issue and it was beginning to look like the problem stemmed from our move functions of the critters. We discovered that the Ants, when eaten, were not properly being removed from memory. This resulted in a segmentation fault that was difficult to resolve. But once we

managed to ensure that the deletion from memory took place, Doodlebugs were finally able to move freely around the board.

Another observation was that our Critters would sometimes move more often that they were supposed to during a single time step. As a result, for one Doodlebug, more than one Ant could be consumed in a single step. To remedy this issue, a team member implemented a private variable to Critter to check if the Critter had already moved during that time step, to prevent extra illegal movement during a time step.
During our testing phase, after entering our unique parameters (since we're doing the extra credit portion), the steps would loop until it got down to zero. However, there isn't much of a point in continuing to loop these steps and having the user press "Enter" over and over again until the program completes, if the board no longer changes before the end. To account for the situations when the board fills with all Ants or Doodlebugs, we would prematurely end the looping, so the simulation can come to an end, since no changes would occur anymore.

This project was technically challenging and coupled with the collaboration aspect, made the project that much more time-consuming than the independent projects done in this class. The main benefit to partaking in the group project was the experience of collaborating without physically being next to each other in a classroom or computer lab. With the distance aspect, we realized the potential need to focus on doling out responsibilities and so that in "off" hours, when we were not all online together, we would know exactly what we needed to be working on. However, in our case, everyone was proactive in contributing to anything that needed to be done at the time, which is not always the case with collaboration projects.

General Work Distribution

Andrew - Critter, Ant, Dbug classes, Board, main.cpp, psuedocode
Henry - debugging, menu class, makefile, Doodlebug class, board class
Joel - Ant, Dbug classes, Critter, menu, board, testing, pseudocode
Frank - debugging, critter class, main.cpp edits, reflection doc, testing, pseudocode