**Group Project Reflection**

**Design**

**General description:**

Problem: Implement 2D Predator-Prey Game with a 20 \* 20 initialized grid of cells and with number of steps determined by the user.

**Design Overview:**

2D Predator-Prey Game is a simulation with specific rules. The preys are ants and the predators are doodlebugs in the simulation.

**Rules & Specific Problems:**

1) Only one critter may occupy a cell at a time.

2) The grid is enclosed and no critter may move off the grid.

3) Time is simulated in steps. Each critter performs some action every time step.

4) The world is initialized with 5 doodlebugs and 100 ants.

5) Ants and doodlebugs are placed randomly on the grid.

6) User enters the number of time steps to run.

**Required functions:**

1) Move() describes ant and doodlebugs movement

2) Breed() described ant and doodlebugs breeding

3) Starve() describes the starvation and death of doodlebugs

**Extra Credit:**

1) User enters the size of the grid rows and columns, the number of ants, and the number of doodlebugs.

**Challenges&Opportunities:**

The most challenging issue, but at the same time the most rewarding and helping issue will be working as a group for the project. Collaboration and cooperation will be the most important opportunities for our group for a well-designed and implemented project. Each of us have provided their own designs for the project. Even though there are differences in the design ideas, we have come up with a general overall design idea to work on.

**Coding Design:**

Each of us worked on the code design. And we`ve come up with a general design as below.

**Main**

Include a menu function **with** options **Play** or **Exit**

**Menu class**

Creates an instance of **Game**

Ask for input on size of grid

Ask for input on number of doodlebugs & ants

Call to the main game function using user inputs

Validate user input

**Board class**

Dynamically creates a 2D board

Prints the state of the board

**Critter class**

Contains a board object

Contains virtual move(), breed() functions for the critter

Contains getters and setter for the type and the location of the object.

**Ant class**

Contains move(), breed() functions for the ant object

**Doodlebug class**

Contains move(), breed(), startve() functions for the doodlebug object

**Game class**

Creates a 2D board

Runs the game with the inputs from the other classes (number of ants, doodlebugs, the

size of the board, steps)

Runs the time steps

Move() function for ants and doodlebugs

Print the board to the screen

Note: We`ve also decided to include functions for EXTRA CREDIT.

**Reflection**

We believe that delegating the tasks in order to create a program that runs efficiently for this assignment was a tremendous learning experience for our team. There were several different strengths each member brought to the team, however effective communication was a common characteristic each individual valued. This took place by following up with our discussion board daily, setting measurable deadlines for tasks to be completed, and all coming into agreement at the beginning what our design should look like. As we entered into the third week of planning this past week, our team even decided to move our comments to another online discussion forum to more easily navigate to questions and comments as needed. Communicating with TA’s for clarity on the instructions and expectations was also an important way for us to maximize the time spent on this program.

Because our team was so thorough with the design of our program our first week of planning, there was not too many areas we ran into that changed our direction of how to code the assignment. The only significant revision we made was that initially, our design had the functions that set the characters on the board defined in the Ant, Critter, and Doodlebug classes. While coding the following week, we mutually decided that it would make more sense to put those functions in the Board class. This would then allow the Doodlebug and Ant classes to call the get and set functions from the Board class when needed instead. We’re proud that each member of our team has given this assignment the time and attention needed to excel. This was shown in our ambition to complete the extra credit portion since we had budgeted our time well from the start. Our team enjoyed this opportunity, and on an individual level, getting the chance to see different styles and thought processes on how to code a common function was a great insight to have moving forward.

**Group Project Test Tables**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case** | **Input Values** | **Driver Functions** | **Expected Outcomes** | **Observed Outcomes** |
| If ant or doodlebug is not adjacent to an unoccupied cell then it will not move | Int x, int y | Int Board::getStatus | Ant/doodlebug does not move | Ant/doodlebug does not move |
| If ant survives three time steps the ant will breed | Int age | Int Ant::getAge | The ant breeds into an empty cell after three time steps | The ant breeds into an empty cell after three time steps |
| Doodlebug will firstly try to move to an adjacent cell containing an ant and eat the ant | Int x, int y | Void Doodlebug::move | The doodlebug eats an ant in an adjacent cell if there is one | The doodlebug eats an ant in an adjacent cell if there is one |
| If a doodlebug survives for 8 steps it will breed a new doodlebug in empty, adjacent cell | Int age | Int Doodlebug::getAge | The doodlebug breeds in an empty adjacent cell | The doodlebug breeds in an empty adjacent cell |
| If a doodlebug has not eaten an ant in 3 time steps it will starve and die | Int age | Void Doodlebug::Starve | The doodlebug starves and is removed from the board | The doodlebug starves and is removed from the board |
|  |  |  |  |  |
|  |  |  |  |  |