Lucus Blanchard

Emily Conry

William Dillon

Ryan Mack

Jessica Speigel

Reflection Document Group 1

CS 162 - Winter 2018

Problem Description

In winter 2018, cs 162, students were tasked with created a predator-prey program. Group One consisted of members: Lucus Blanchard, Emily Conry, William Dillon, Ryan Mack and Jessica Speigel.

Predator-Prey Program

* Consists of two insects - Ants and DoodleBugs
* They bugs live in in an enclosed 20 x 20 grid
  + Unless the extra credit option is taken (detailed below(
* One one doodlebug can operate one space at a time
* Ant behavior (char “O”)
  + Move - up, down, left right
  + Breed - create an ant after 3 steps, one 1 per ant per 3 steps
* DoodleBug behavior (char “X”)
  + Move - towards an ant to eat him (adjacent cell)
  + Breed - after 8 steps creates a new doodlebug, once per db per 8 steps
  + Starve - must eat one ant per 3 steps or it dies
* Gameplay
  + 5 ants and 100 doodlebugs are initialized
  + Gather user input to steps (time)
  + Doodlebugs move before the ants
  + Loop program
* Extra Credit
  + Gather user input for : size of the grid row/columns && # of ants && #doodlebugs

Work Distribution

Group One was set up early on and had great group participation. Members were active on the discussion forums, posted announcements, submitted code, and ideas that contributed to the group project. Our group started on canvas but the group found the canvas system to be clunky for our purposes. The group saw the need for a project github as to protect the master files for our program while still allowing everyone to push code. Jessica S. created this for the group. She helped us maintain version control as our github gatekeeper and also contributed to our codebase.

We also thought there was a need to have a different file management system so a team google drive was created by Ryan M. Ryan M. also contributed to a lot of the code. For example, he fixed most of our compilation errors and did extensive testing of our program. To give our team a place to converse in real time a slack channel was created by Emily C. She also worked on the reflection document, the ascii art (beginning and end) and the file reader.

Aside from greatly contributing to our code Lucus B. really helped to keep our group on schedule and on task. William(Brad) did some amazing work on the staving functions as well as ...

Original Design

Originally our group brainstormed on the announcement and discussion sections of canvas. We debated how many classes we would need and initially had thought of 7+, but thanks to great group input we narrowed down the list of classes to 5. A Grid, Critter, Menu, Ant and Doodlebug class. The group decided early on that we would definitely be doing the extra credit for this project. The extra credit required us to request user input as to how many doodlebugs and how many ants would exists on the user created gird. We also experimented with several different input validation techniques prior to deciding upon using this:

ex. while (cin.fail()|| (choice < 1||choice > menuItems.size()))

{ cin.clear();cin.ignore(999,'\n');cout << "I'm sorry, that's not a valid choice.

Please enter a choice between 1 and " << menuItems.size() << endl;cin >> choice;}

Problems and How We Solved Them

During this project our group had a few coding issues that we needed to work out. At one point our coding we were unsure whether the board state was saved at the end of the simulation of if the board reset. The requirements detailed that the current state of the board must be maintained at the end of the steps. Several of our teammates discussed this on our slack channel and a question was posted to piazza. Our instructor answered that the simulation at end should prompt the user to 1. Run the simulation again or 2. Exit. If “Run the Simulation Again” was chosen the user would be prompted to enter steps again. And a new grid would not be created. Our team properly designed the grid with this new information.

We also weren’t sure about the breeding and starving behavior for doodlebugs on a single turn. It was determined that this was a design decision and our code was modified to reflect this. Our final issue revolved around memory leak issues. Specifically, we encountered a phenomena that made valgrind report:

HEAP SUMMARY:

==24354== in use at exit: 32 bytes in 1 blocks

==24354== total heap usage: 148 allocs, 147 frees, 33,122 bytes allocated

==24354==

==24354== Searching for pointers to 1 not-freed blocks

==24354== Checked 184,200 byte

We fixed this by

Testing Table and Results

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case | Location | Input Values | Expected Output | Observed Output |
| Play game or  Quit | menu.cpp/.hpp | (-765, -10900, 6785, t, trixie, <\*\*\*>, ~!@)  () | (all errors)  (plays the game or quits) | ()  () |
| User Input:  Size of grid | menu.cpp/.hpp | (-11, -2036, y, bianca, ??, ./, 1)  (7, 2356) | (all errors)  (creates grid of appropriate size) | ()  () |
| Ant Display character ‘’ |  |  |  |  |
| Doodlebug Display character ‘’ |  |  |  |  |
| User Input:  # of Ants | menu.cpp/.hpp | (-56, -1717, n, ben, ??, &\*,)  (5 13, 3766) | ()  () | ()  () |
| User Input  #of doodlebugs | menu.cpp/.hpp | (-4, -2222, x, shangela,!!, #@, )  (2 225, 2567) | ()  () | ()  () |
| User Input  # of steps | menu.cpp/.hpp | (-17, -95533, jk, milk, ::, %^&%$, )  (4, 45, 309666) | ()  () | ()  () |
| Ant Movement |  |  | Ant moves up, down, left or right randomly |  |
| Ant Movement |  |  | If cell is occupied Ant stay where he is |  |
| Ant Breeding |  |  | Every 3 steps ant breeds, no more than once every 3 steps per ant |  |
| Doodlebug  Movement |  |  | Move to adjacent cell to attempt eating of ant. If ant is not eaten moves randomly up, down, left, right |  |
| DoodleBug Breeding |  |  | Every 8 steps a doodlebug breeds, not more than once every 8 steps per doodlebug |  |
| Doodlebug Starving |  |  | Doodlebug dies if he hasn’t eaten in 3 steps |  |
| Grid Enclosure  check |  |  | When Ant or Doodlebug reaches the end of the grid they should |  |