Lucas Blanchard

Emily Conry

William (Brad) 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: Lucas Blanchard, Emily Conry, William (Brad) Dillon, Ryan Mack, and Jessica Speigel.

Predator-Prey Program Requirements

- Consists of two insects - ants and doodlebugs
- The bugs live in in an enclosed 20 x 20 grid
    - Unless the extra credit option is taken (detailed below)
- One insect can occupy one space at a time
- Ant behavior (char “O”)
    - Move - up, down, left right
    - Breed - create an ant (if an empty square exists) after 3 steps, one 1 per ant per 3 steps
- Doodlebug behavior (char “X”)
    - Move - towards an ant to eat him (adjacent cell)
    - Breed - creates a new doodlebug after 8 steps (if an empty adjacent square exists), once per doodlebug 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 in rows and columns, number of ants, and and number of 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 real time collaboration. The group saw the need for a Github repository for the project 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 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 and created ascii art to introduce the game and say goodbye to the user.

Aside from greatly contributing to our code Lucas B. really helped to keep our group on schedule and on task. Brad D. did some amazing work on the starving functions as well as added descriptions for numerous functions throughout the program files. Also, Brad raised some interesting questions about the design of the program and posted the questions on Piazza.

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 excellent 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 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 during 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. One TA 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 designed the grid with this additional information. Later on, another TA answered that the “Run the Simulation Again” option was chosen, the game should completely reset. We decided as a team that we would include both options since it did make sense within the program flow. When the simulation runs out of steps, we provide the options to “Continue running the simulation” or “Stop simulation”. The first option allows the user to enter an additional number of steps to run the simulation for. If they choose the second option, the main menu is displayed that allows them to start a new game or exit the program.

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.

Conclusion

This project went very smoothly for being so complicated. The usage of Github for source control and Slack for day-to-day collaboration were extremely helpful and helped us work together simultaneously. Everyone in the group was extremely collaborative and responsive.

Testing Table and Results

Test Case	Location	Input Values	Expected Output	Observed Output
1. Run a predator/prey simulation 2. Exit	menu.cpp /.hpp	(-765, -10900, 6785, t, trixie, <***>, ~!@) (1, 2)	(all errors) (plays the game or quits)	(All errors as expected) Int 1 plays the simulation and 2 quits as expected)
User Input: Steps to run simulation for	menu.cpp /.hpp	(thanks, -2, 0, thanks) (1, 20, 599)	(all errors) (sets number of steps to run simulation for)	(All errors as expected) (steps run appropriately once the game starts)
User Input: Size of grid cols/rows	menu.cpp /.hpp	(-11, -2036, y, bianca, ??, ./, 7, 55) (25, 35, 47)	(all errors) (creates grid of appropriate size)	(All errors as expected) (grid created appropriately)
Ant Display character ‘O’			Ants displayed correctly	Ants are displayed correctly
Doodlebug Display character ‘X’			Doodlebug displayed Correctly	Doodlebugs are being displayed correctly

User Input: # of Ants	menu.cpp /.hpp	(-56, -1717, n, ben, ??, &*,) (5 13, 25, 77)	(All errors) (appropriate number of ants created)	(All errors as expected) (appropriate number of ants created)
User Input #of doodlebugs	menu.cpp /.hpp	(-4, -2222, x, shangela,!!, #@, 225, 2567) (2 , 5, 9)	(All errors) (appropriate number of doodlebugs created)	(All errors as expected) (appropriate number of doodlebugs created)
User Input # of steps	menu.cpp /.hpp	(-17, -95533, jk, milk,::, %^&%$, 309666)) (4,900)	(All errors) (4 steps, 900 steps successful)	(All errors as expected) (steps function successful)
Ant Movement			Ant moves up, down, left or right randomly	Ant is behaving as expected
Ant Movement			If cell is occupied Ant stay where he is	Ant is behaving as expected
Ant Breeding			Every 3 steps ant breeds, no more than once every 3 steps per ant	Ant is behaving as expected
Doodlebug Movement			Move to adjacent cell to attempt eating of ant. If ant is not eaten moves randomly up, down, left, right	Doodlebug is behaving as expected
Doodlebug			Every 8 steps a	Doodlebug is

Breeding			doodlebug breeds, not more than once every 8 steps per doodlebug	behaving as expected
Doodlebug Starving			Doodlebug dies if he hasn't eaten in 3 steps	Doodlebug is behaving as expected
Grid Enclosure check			When Ant or Doodlebug reaches the end of the grid they should	Grid is solid – no bugs escape this day
The simulation runs for the correct number of steps			The number of steps at the end step matches the number the user entered when starting the simulation.	Correct number of steps are run.
After the simulation is complete, the appropriate menu appears: 1. Continue running simulation 2. Stop simulation	menu.cpp /.hpp	(-765, -10900, 6785, t, trixie, <***>, ~!@) (1, 2)	(all errors) (plays the game or quits)	(All errors as expected) Int 1 prompts the user for additional steps and 2 brings up the main menu as expected)
The user is prompted for additional steps			The same steps prompt from the beginning of the game appears.	The user is prompted for input.
The program allocates and deallocates memory correctly.			No memory leaks.	Valgrind reports no leaks are possible.