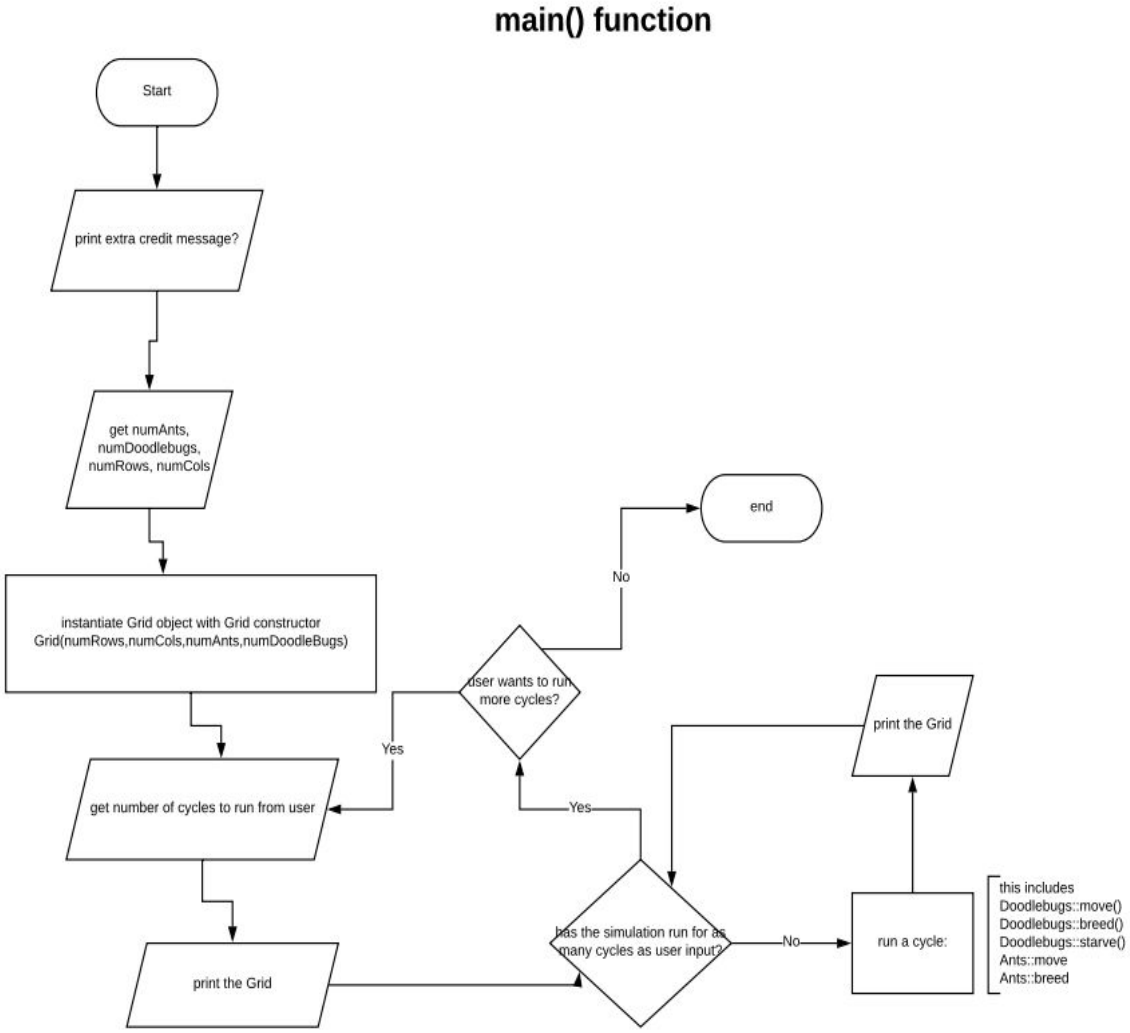
Group Names: Alex Swider. Alex Stanley, Jordan Bartos, Gabriel Rice, Arthur Rodrigues

Date: May 13, 2018

Reflection

The Predator Prey group project was different from what we've encountered so far in the course. While many projects have their own unique challenges, one of the primary challenges in this project was being able to effectively communicate. In addition, another challenge we encountered was being able to read and understand different styles of code, as we each had our own unique styles of writing. Our first method of getting started on the project involved using canvas as a platform for communication. This led to the scheduling of two meetings that were held via google hangouts. During the meeting, work was delegated to group members. Namely, the Critter class was assigned to Alex Stanley, Ant class to Gabriel, Doodlebug to Arthur and Alex, and Grid to Jordan. Alex Swider was unable to attend the meeting for personal reasons. Alex worked with Jordan in writing the main function and reflection documents. With the assignments in mind, everyone worked independently on their portions to start the project, routinely checking in on canvas and using github as a platform for sharing the code we wrote. After everything was written, it took much more teamwork within the group to ensure variables, functions, and classes worked together to run the program.

In tackling the project, we found it would be best to revisit old projects we worked on and use old code to expedite the writing process. In addition, we found it helpful to use flowcharts and diagrams to illustrate the flow of the program. The following page contains a graphic made by Jordan outlining the intended flow of the program. This graphic was helpful when designing the main function and thinking about what would be needed in the various class functions.



This was a great starting point, but then of course each person had their own techniques for planning out how to write their code. After the classes were written and uploaded to github, it was possible to dive into writing the main function and work our way toward testing the simulation. A number of tests were run on the program. The first tests included ensuring basic function of the program. Afterword, tests designed for special cases were implemented, such as when Doodlebugs and Ants are at the edges of the board. After ensuring our various tests worked (more detailed test breakdown located on chart on the following page), we focused our efforts on ensuring proper memory management.

Test Case	Input Value	Driver Functions	Expected Outcome	Observed Outcome
Inputs too low	Inputs < 0	-Class Grid func's -Class Ant func's -Class Doodlebug func's	Prompt user input is invalid and to redo the input	User prompted to enter values again, program does not start without valid input
Inputs in proper range	Inputs > 0 but less than extreme values	-Class Grid func's -Class Ant func's -Class Doodlebug func's	Functions take in the values and run the program accordingly	Program works properly.
Inputs in proper range	Inputs at high extremes	-Class Grid func's -Class Ant func's -Class Doodlebug func's	Functions see values too high and prompt for reinput	Program works properly (though takes a while to execute and print).
Inputs too high	Inputs too high and above upper limits	-Class Grid func's -Class Ant func's -Class Doodlebug func's	Functions see values too high and prompt for reinput	User prompted to enter values again, program does not start without valid input
Input not of proper data type (character)	Inputs are letters when requesting integer	-Class Grid func's -Class Ant func's -Class Doodlebug func's	Function cancels this input and prompts for re-entry	User prompted to enter values again, program does not start without valid input
Doodlebugs and Ants near edges	n/a	-Class Grid func's -Class Ant func's -Class Doodlebug func's	Program still functions properly	Functions behave properly, both stay within the bounds of the field.

As far as other tests, we ran specific numbers to verify the proper function of different aspects of different classes. In particular, we wanted to test: the doodlebugs ability to eat an ant (and not a doodlebug), the doodlebug's starvation mechanic, the doodlebug's movement mechanic (ensuring it moves toward an ant if nearby), the breeding mechanic for both classes (and that the breed counter works appropriately), and finally that the ant is behaving properly in moving as well.

Function	Test method	Desired behavior?
Doodlebug eating	Input lots of doodlebugs with a few ants on smaller board, observe behavior	Passed test, doodlebugs did not eat other doodlebugs, and ate ants when near them
Doodlebug movement	Same as for eating method	Recognized when ant next to doodlebug, otherwise random movement. Responded well to edge cases as well.
Doodlebug starvation	Lots of doodlebugs, 1 ant, observe what happens after 3 moves	Passed, died after 3 moves for the most part (unless one was next to the ant, in which case it survived longer)
Doodlebug breeding	Healthy amount of doodles and ants	Breeding mechanic worked appropriately
Ant Movement	Regular amount of doodles and ants	Ant moved appropriately. Responded well to edge cases.
Ant Breeding	Regular amount of doodles and ants	Ants bred at their proper intervals.

Moving forward, we learned some strategies for tackling group projects better in the future. Two key aspects are to communicate often and early. This will ensure each member has an ample amount of time to complete their portion of the project. Even more important to starting early is being able to merge code written by different people. Code may be written with different variables or in different formats, and this can be troublesome when assembling the code into the final program. Finally, early communication can ensure extra time when debugging, which is critical for projects of this magnitude.