Module B

Goals-

Design, write, compile, and test a program that uses functional decomposition to modify your program to have significantly different types of movement
Develop a simple test plan

This module builds on the previous module. You will receive feedback as soon as possible. **Save a copy of the previous module** so you have a known place to start in case of catastrophe! Design your changes. Review the design. Then, and only then, start coding!

You will modify your program so that the user can specify the number of Critters. You will need to change the array to be an array of pointers to Critter objects. This may require minor changes to your move and/or relocation functions.

You will need to use a pointer indicate blank cell in the array. How will you do this? You can use a NULL pointer. Another option might be to somehow modify the Critter class to indicate if the object is a Critter or no_Critter.

With more than one critter you do not stop the simulation when a critter tries to move off the array. Instead if the chosen direction would move the critter off the array, don't move it.

You can move through the array row by row to find the Critters to move. The random move of a Critter may put it in a position to be moved a again in the same "generation". Once option to avoid this problem is to use a variable to mark if it's been moved. If so, then you skip it. You can then go through the to print it to the display. Remember to also reset the variable marking a move.

Design your changes BEFORE writing the code. If you do this correctly, AND test it the next part is almost trivial.

You will prompt the user for the number of creatures. You will **create that many Critter objects** and randomly insert them in the array. Display the starting positions. Then move all the Critters. If a Critter would move onto another Critter or off the array then it stays in place. Otherwise the movement of each Critter should not be different. After you have determined the new location of all the Critters display the array. Repeat until the simulation is completed.

You must validate your input! Beyond the usual, "did they enter a number when I wanted a number" you now must check if the user requested more critters than cells in the array. Is there anything else?

How do you stop your program? Do you prompt the user after each move (could get tedious)? Do you prompt them for a number of steps when starting the simulation? Something else?

Finally, you should include all 6 files into a zip archive. You must submit only the zip file in TEACH by the due date. We will only accept submissions in TEACH that are in a zip file and that include the makefile.

Grading

Modules will be graded S/U. As part of the cumulative modular program (Cump) this module will contribute to that numeric score. These criteria are given for reference.

Programming style- 10%

You create the number of Critter objects specified by the user- 20%

Your array correctly holds more than one Critter object- 20%

All Critters move correctly, i.e. not onto each other, nor off the edge- 20%

All input is validated correctly - 20%

Your program exits correctly- 10%