Thomas Buteau
CS162-400
1-22-17

Project overview:
This project is to write a simulation of Langton's Ant. The user will be prompted to decide the number of steps the simulation runs for, the dimensions of the simulation grid, and the user will also either choose a starting location in the grid for the ant or have one picked at random. These choices will be tested to ensure they are valid. The simulation itself is of an "ant" on a grid of cells that can either be white or black where the ant turns and moves based on the color of the cell it occupies, changing the cell colors as it goes. The program ends when either the number of steps that the user defined is reached or the ant tried to move out of the bounds of the grid.

Program design:
The program will be written in three main sections: Menu function, Board Class, and Ant Class.

Menu Function
The menu function will start a while loop. In that loop the user will be presented with introductory text, the options available to create the simulation board, determine the maximum number of steps, start the simulation, or exit the program. The menu function will call on a data validation function to verify the user input is correct before acting on said input. After the user enters all the necessary information and tell the simulation to run the menu function will generate the Board and Ant objects and begin the simulation.

Board Class
The board class takes in two integers as arguments, a row length, a column length. It uses the row and column arguments to dynamically generate a 2D array for the simulation board. It has a set method to change the value at specified locations on the array. It has a method to display the board array on the screen as well as the number of steps taken and, if the simulation has ended, a message saying the reason simulation has finished. It has a method to return true or false for if a location is on the board. The deconstructor for this class will deallocate the memory used by the board dynamic array.

Ant Class
The ant class will take in a starting location, the maximum number of steps, and a Board object as arguments. It will have a current location variable which holds its position in the board array, a future location variable which holds the position of where the ant will move to, a direction variable, a variable for the maximum number of steps, a variable for the current number of steps, a variable for if the game should continue, a variable for the current array location's color, and a variable for the future location's color. It will have a method that changes the current direction based on the current location's color, a method to change the current location's color, a method that checks if the future location is in the board array, a method to check if the end game conditions have been met (maximum number of steps reached or board edge is reached). Finally, there will be a method to move which will utilize the other methods to change directions, verify the future location is inside the grid, verify the number of steps has not reached the maximum, updates the number of steps, changes the current location's color, updates the future location color, moves the Ant to the future location (assuming the end conditions are not met), updates the current location color variable, and calls the Board draw method to print out the new

array. The deconstructor will deallocate the memory the Ant object uses when the simulation is finished and returns to the menu.

Test Plan

To test this program each option in the menu will be attempted with a non-integer input, an integer input under the acceptable range, an integer input over the acceptable range, and an integer input in the acceptable range. Likewise, each sub-option in the menu function will be similarly tested. The table below shows a general summary of the testing methods. Note that the table is not a complete list of tests used.

Test Case	User Input	Test Location	Expected Outcome	Observed Outcome
Non-integer input	"x"	Menu Function	Error message	Error message
Integer out of range	50	Menu Function or sub-functions	Out of range message	Out of range message
Valid input	6	Menu Function	Exits program	Exits program
Valid input at incorrect time	5	Menu Function	Required data error message	Required data error message

Post Build Reflections

Several pieces of the general design were changed while writing the code. For example, originally the Board class was going to have a constructor that took several arguments and generated the 2D array at object construction. While testing the Board class this worked fine, however, when the Ant class was built with a Board object as an argument in the Ant class's constructor error messages appeared about the Board class not having a default constructor. After some troubleshooting on this I decided it would simply be easier to give the Board class a default constructor with no arguments and get/set methods to modify its variables as well as a method to generate the 2D grid array.

Another change in the design was the implementation of an integer verification function. The menu function is already longer than it should be and adding a separate function to validate user input roughly cut the line length of the menu function in half.

The move method for the Ant class was also doing too much and was too long so the bulk of its tasks were made into functions themselves and now mostly just calls different functions in a while loop.

This assignment has shown me that while a design will likely not survive implementation completely intact it does save a great deal of time by reducing rework.