Alex Rueb
Oregon State University
CS162-400
4/16/2017

Project 1: Design, Testing and Reflection

Problem Description

Develop a simulation of Langton's Ant, a construct developed by Michael Langton in 1986 where an ant is placed on a grid of black and white cells and must follow a simple set of rules. If the ant is occupying a white square, it must turn right, flip the cell to black, and then advance one square. If the ant is on a black square, it must turn left, flip the cell to white, and then advance one square.

In this particular programming assignment, the user needs to be able to enter the size of the grid/board in rows and columns, specify the number of steps the ant should perform, and specify the starting position of the ant. The board needs to be displayed to the user for each of the specified steps. The program must also be able to handle the event in which the ant encounters a boundary of the board so that it can successfully complete the required number of steps requested by the user. This behavior is to be programmed at the discretion of the designer. The user should also have an option to use a randomly selected start position for the ant that falls within the specified board size.

Design Solution:

Based on the requirements of the program, my design solution will consist of two primary classes to run the simulation: an *Ant* class and a *Board* class. As the name suggests, the Ant class will represent the ant and include data members to hold the color of the cell the ant is occupying and the direction the ant is pointing. These values will be enumerated values since they will be used throughout the program and will only consist of a set number of possible values. The Ant class will also include corresponding "set" and "get" methods for each data member.

The Board class will contain fields for the number of rows and columns of the board, the x and y coordinates of the ant, the number of steps to be performed by the ant. These values will be set initially via a constructor defined for the class upon creation of a Board object. The constructor will use these values to dynamically create

a two-dimensional array that will serve as the board that the ant will traverse. Since this creating the board will dynamically, the memory allocated to the array will need to be freed by a destructor. Last but not least, the Board class will contain an object of the Ant class previously described as a data member.

The majority of the functionality of the program will be included within the ant class. A member function will be provided to move the ant, based on the rules described in the problem description (see pseudocode below). If a move will require the ant to go out of bounds, it will deviate from the standard ant behavior by turning 180 degrees and proceeding back in the opposite direction. The Board class will also include a member function to print the board, since this will need to happen for each step. Lastly, there will need to be a member function that creates a loop to call the move function and perform the number of steps specified by the user. There will likely be some logical supporting functions that present themselves as well to minimize redundant use of code and improve the clarity/readability of the logic.

Pseudocode to Move the Ant (member function of Board class):

Get the color of the cell that the ant is occupying from the ant object
Set the cell color on the board to the opposite of the ant object color
Get the which direction the ant is facing from the ant object
Determine where the ant will move from cell color and direction (up, down, left, right)
Check if the move will place the ant out of bounds
    If the ant will go out of bounds:
        Set X and Y coordinates to move ant in opposite direction
    If the ant will not go out of bounds:
        Set X and Y coordinates to move the ant as planned
Set the board color at the XY coordinates to the ant object for next move
Change the board array to ant character

Testing Plan:

Input(s)	**Expected Output**	**Observed Output**
Ant occupies: White Cell Ant Pointing: Up	Ant moves east Cell changes to black	Ant moves east Cell changes to black

Ant occupies: Black Cell Ant Pointing: Up	Ant moves west Cell changes to white	Ant moves west Cell changes to black
Ant occupies: Black Cell Ant Pointing: Down	Ant moves east Cell changes to white	Ant moves east Cell changes to white
Ant occupies: White Cell Ant Pointing: Down	Ant moves west Cell changes to black	Ant moves west Cell changes to black
Ant occupies: White Cell Ant Pointing: Right	Ant moves south Cell changes to black	Ant moves south Cell changes to black
Ant occupies: Black Cell at top boundary of grid Ant Pointing: Right	Ant moves south Cell changes to White	Ant moves south Cell changes to White
Ant occupies: White Cell at bottom boundary of grid Ant Pointing: Right	Ant moves north Cell changes to black	Ant moves north Cell changes to black

Reflection

When I initially sat down to tackle this assignment, my initial thought was that this project seemed to complicated to sit down work through without writing any code. In fact, it seemed like it would be easier to just start writing code until a plan materialized. However, once I started writing things down and taking things step by step, it was clear that taking the time to plan the program before writing code was definitely time well spent.

As soon as I started listing all of the fields that would be needed to create the board and control the ant, it became relatively easy to arrange them into groups and form the class structure. Visualizing the behavior of the ant did prove to be tricky, but I found writing down each step as pseudocode helped me keep my place in the logic and work through the steps to cover both the standard ant behavior and the even that the ant would go out of bounds.

After organizing the classes, data members, and logic required to build the necessary functionality, I felt I had a much clearer understanding of the problem and was ready to write the code. With a plan in place, writing the code went smoothly and everything fell right into place for the most part. As soon as I started testing my code, I decided it would be helpful to modify the ant marker so that it gave a visual indication of the direction the ant was pointing. I found the <, ^, >, v characters to fit the bill perfectly for this purpose.

In order to add the directional ant markers, I have to make some changes that were not included in my original code, but it ended up being a fairly minor modification. I also created some helper functions to streamline bits of code that were used repetitively. Another modification/improvement worth mentioning is that when I first started writing my move function in the Board class, I had eight different blocks of if statements to handle the various color/direction pairs that would determine where the ant would move next. I soon realized that each block very similar to another of the eight blocks and could be condensed into four blocks if I used the logical OR operator in the if clause.

All in all, I found this assignment to be very satisfying and I can definitely seeing the value in having a plan before sitting down to write the code. I will be making an effort to continue developing and improving this skill throughout this course and beyond.