David Mednikov

CS 162 W17

onid: mednikod

# Project 1 Reflection

## Project Plan

My initial project plan consisted of jotting down ideas that popped into my mind while trying to break this assignment down into smaller parts. First, I had to make a 2D array. Each element of the array would represent the color of that cell. I decided to make it an array of integers instead of enums. A value of 0 meant a white cell, 1 meant a black cell, and 2 meant the ant. I had to make an ant class, with 2 member variables: the color of the ant's square (an integer) and the direction the ant is facing (an enum). The inputs would be the number of rows, the number of columns, and the number of steps. I would add a menu and the option to select the ant's location later. The output would be to print the board that the ant is on after each step. I had to decide what behavior to use when the ant hit the edge. I decided to make the ant wrap around. Before I had looked at the grading requirements, I decided to create a getInt() method that would validate a user's input and make sure they entered an integer. I knew I had to have a moveAnt() method that would move the ant, change its position, and change the color of the square it was on. That was the extent of my plan early on.

## Test Plan

| Test Case | Input Values | Driver Functions | Expected Outcomes | Observed Outcomes |
|---|---|---|---|---|
| Negative input for # of rows | Input < 0 | main() calls recursive function to call getInt(0) | Loop back and prompt user for positive input | Loop back and prompt user for positive input |
| Input = 0 for # of steps | Input = 0 | main() calls recursive function to call getInt(0) | main()<br>Ant makes 0 moves | main()<br>Ant makes 0 moves |
| Input for ant's location higher than # of rows | Input > rows | main() calls recursive function to call getInt(lowNum, highNum) | Loop back and prompt user to enter a number in range | Loop back and prompt user to enter a number in range |
| Input for # of rows in correct range | 0 < input< 50 | main() calls recursive function, call getInt(1, 50) | Accept user input, set rows variable | Accept user input, set rows variable |
| Input contains letter | 25a | main() calls getInt(), and cin.gcount() > 1 | Loop back, prompt user to enter integer | Loop back, prompt user to enter integer |
| Input is a decimal | 50.5 | main() calls getInt(), and cin.gcount() > 1 | Loop back, prompt user to enter integer | Loop back, prompt user to enter integer |
| Test Ant | Steps = 52 | getInt(0) | Ant makes a heart shape | Ant makes a heart shape |

## Test Results

All of my tests passed. I got the expected outcome for all of them. Using cin.fail() and cin.gcount(), I was able to ensure that my getInt() method only returned an integer within the specified range. I used getInt() to get the user's menu choice, number of rows and columns, ant's starting location, and the number of steps. Each use of getInt() has its own parameters.

Also, while reading about Langston's Ant online, I stumbled upon a website that showed some patterns that the ant makes. At 52 steps, the ant is supposed to make a heart shape. This was how I tested the success of my actual ant. If the ant drew a heart after 52 steps, then my ant is following the rules correctly.

## Comments

The first issue I ran into while coding was using the array. I was thinking about it wrong. I was treating the 2D array like a graph. As in, I wanted to represent the ant's location like this: board[xCoord][yCoord]. However, I quickly found out that this would not be possible, as the 1st dimension of the array makes up the rows of the board and the 2nd dimension is the columns. I had to re-write some of my code to adjust to my change in how I referred to the array elements.

Another issue I came across was printing each turn while also remembering the color of the square that the ant was on. I had to use a similar method to swapping numbers, i.e. a temporary variable to remember the color of the square, make the square an ant, print the board, then change the color of the square back to the temporary variable.

I also had to find a way to make the ant wrap around the board when it hits the edge. At first I was unsure how to do this, and was planning on making the ant just turn around. But I realized that making the ant wrap around made more sense for this program. I had to think about the column just to the right of the right edge as being the left-most column (and the same for the other 3 edges). It was essentially a round board that appeared as a rectangle. Once I figured out that the ant only crosses an edge in certain situations, I was able to use if statements to move the ant to the other edge of the board.

It also took me a little while to get the Makefile working. I realized that I was forgetting to include some header files in my object recipes. Once I figured that out the Makefile was easy and very useful.