Romano Garza
05/16/17
CS-162-400

Project 1 Reflection

Pseudo code design:

```
Main-
Create variables to store rows, columns, ant row, ant column, number of moves ant will make,
and ~~a bool variable that tells if the ant will be randomly located~~ set stored variable equal to less
than 0 value.

Introduce program

Display menu of choices
        -choice one: start program
        -choice two: fill in size specifications for matrix
        -choice three:  enter number of steps ant will take
        -choice four: enter ant starting location
        -choice five: quit the program
Get choice

If user chooses one but matrix size or number of steps have not been specified
        Display error message
        Start at display menu again
Else, if user hasn't specified a starting location
        Let them know it will be set randomly
        Process their request given the rows, columns, ant row, ant column, moves ~~and ant
        random variable~~
        Display menu again
Else,
        Process their request given the rows, columns, ant row, ant column, moves
        Display menu again
If user chooses two
        Get size of matrix (rows and columns)
        Validate input (user must enter numbers >0 and <=80)
        Display menu again

If user chooses three
        Get number of steps
        Validate input (user must enter a number >0)
        Display menu again

If user chooses four
        First make sure user has specified the size of their 2d array
        Get ant starting location (ant row and ant column)
        Display menu again

If user chooses five
        Quit

Processing Request-
Build the 2d array according to size specifications
Write the array with random 0's and 1's

If ~~ant is random~~ user hasn't entered an ant row or ant column
        Start the ant at a random location ~~and set the ant moving program in motion given the
        board created, row and column size, as well as number of moves~~
Else
        Set the ant moving program in motion given the board created, row, and column size,
        ant location, as well as number of moves

Randomly write 0's and 1'values to the matrix locations via srand and rand()%2 which will
either = 0 or 1. Thus, the board is constructed randomly with each spot having an evenly
weighted chance of being a 0 or 1.

Start ant out randomly-
Randomly select a number between 0 and one less than the number of rows and set the ant
column location to this number

Randomly select a number between 0 and one less than the number of rows and set the ant
column location to this number

Set the ant moving program in motion given the board created, row and column size, ant
location as well as number of moves

Move the ant-
Set ant direction to the top of matrix (north ~~or top~~)

While the ant hasn't moved as many times as specified
        Print the matrix
        If the ant is at a 0
                Switch the 0 to 1
                Depending on where the ant is facing make it's new direction to the right
                (clockwise)
                Move it to the ensuing column or row
        If the ant is at a 1
                Switch the 1 to 0
                Depending on where the ant is facing make it's new direction to the left (counter
                clockwise once)
                Move it to the ensuing column or row
```

Reflection:

Most of the initial changes to my design came to the area of moving the ant, which makes sense given that it was the part of my design I was most general in. I had an idea of how to orient the ant. However, I did not initially have a failsafe way of implementing a plan to move the ant without it going out of bounds, thus causing my program to run improperly. So while writing the code, I tested out some different processes and I found a simple solution to moving the ant east and south by adding one to the ant row or column and finding the remainder when dividing by total columns, this would move my ant properly south to greater rows or east to greater columns and in both cases eventually back to 0 if it reached the outer limits. However, the problems were in going the other direction, west one column or north one row. I couldn't simply subtract one from the ant column or row and then proceed in the previously mentioned way. So, I just wrote an if/else statements to prevent the ant from going out of bounds and causing program errors. This area of my program required the most attention and debugging as it was rearranged and modified the most.

I ran into a lot of strange issues when I renamed and deleted files in my project environment on Xcode. This was due to redesigning the layout of my files and functions. Eventually a file was missing, I couldn't figure out the issue with the project so I simply copied and pasted my code into a new project document and deleted all the old files. In the future, I'll likely plan for such disorganized file outcomes by nesting my files and functions in an initially more cohesive manor.

In my input validation, I was running into errors where my input wasn't being properly validated. This turned out to be because I was using 'and' instead of 'or'. The problem was that when I wrote the and/or statements for the input validation, I heard in my head what should be correct (all of them) and not what the code should be checking (if only one is wrong …). This little mix up has been noted and will be remembered in future settings.

There were plenty of other issues. For example, some of my functions were being passed a lot of different arguments. And a mix up occurred that took some time to find. This was because I made some simple error in a couple of my function calls, passing arguments out of order, which required amending. I also needed some help with the use of srand() and rand() in my random assignments, but I found what I needed to know by researching online and in the book.

What I learned during this assignment is that attacking a problem in an organized manor is essential to writing code and debugging quickly. If the design is not thoroughly laid out before writing the code, it is possible the code will need to be rearranged and give more room for bugs to be introduced. I also learned that when drafting a design perhaps it is better to be more specific than general, as being general can lead to, again more rearranging and debugging than if one were to be specific in their draft.