Project 1 Design
Michael Nutt
CS162

Problem: Create a Langton's Ant simulation. Given a MxN board filled with white squares, an ant is placed at some point (M,N) from which it will begin moving according to the following rules:

- If the ant is on a white space, turn 90 degrees and change the space to black.
- If the ant is on a black space, turn 90 degrees and change the space to white.

LangtonAnt main function:

1. Execute the menu function.
2. Prompt the user for the following:
    a. Number of rows in the board (rowCount).
    b. Number of columns in the board (colCount).
    c. Number of steps during the simulation (totalSteps).
    d. If the user would like a random starting point.
        i. Generate a random starting row and column.
    e. If the user decides not to use a random starting point:
        i. The starting row of the ant (currentRow).
        ii. The starting column of the ant (currentCol).
3. Generate the 2D array to represent the board.
4. Generate a Board object using the 2D array, colCount, and rowCount.
5. Generate an Ant object using the Board object, currentRow, and currentCol.
6. While currentStep < totalSteps
    a. Increment currentStep.
    b. Print the board.
    c. Check if the ant can move from the starting location
        i. If the ant can move, move the ant in the direction it is facing and toggle the color of the space from which it came.
        ii. If the ant cannot move, reverse the ant's direction and move/toggle.
    d. Prompt the user to either run the program again or quit.
7. Deallocate memory.

return 0.

Menu function:

Prompt the user to either start the simulation or quit. If the user selects "Quit", return 0.

Project 1 Design
Michael Nutt
CS162

Ant class:

Properties:

- int currentRow: The row on which the ant currently sits with a getter.
- int currentCol: The column on which the ant currently sits with a getter.
- char direction: The direction the ant currently faces (north, south, east, or west) with a getter.
- Board *board: Pointer to the board object. Used to track the state of the board.

Methods:

- int getCurrentRow: Returns the row on which the ant currently sits.
- int getCurrentCol: Returns the column on which the ant currently sits.
- void turnOnBlack: Turns the ant 90 degrees to the left.
- void turnOnWhite: Turns the ant 90 degrees to the right.
- void moveAnt: Moves the ant to the next space after turning and checking if the move is in bounds.
- void reverseDirection: Turns the and 180 degrees in the case that it cannot move due to its current location and direction.

Board class:

Properties:

- int rowCount: Number of rows set by user with getter.
- int colCount: Number of columns set by user with getter.
- char **boardTracker: 2D array with rowCount-rows and colCount-columns.
    - https://stackoverflow.com/questions/1533687/write-a-class-using-a-two-dimensional-dynamic-array

Methods:

- int getRowCount: Method returns the number of rows on the board.
- int getColCount: Method returns the number of columns on the board.
- char getCurrentColor: Method returns the color of the given space.
- void toggleSpace: Method to toggle the color of a space visited by the ant.
    - If the space is white, switch to black (#).
    - If the space is black (#), switch to white.
- bool isInBounds: Method to check if the next move is within the bounds of the board.

Project 1 Design
Michael Nutt
CS162

InputValidation:

isIntegerInput function: Takes user input as a string. Returns true if the input is an integer value and returns false otherwise.

convertInputToInt function: Calls the isIntegerInput function to determine whether the input is an integer value, then converts it to an int type and returns that value.

RandomStartGenerator:

selectRandomStart function: Prompts the user to either use a random starting point or to select their own starting point. Returns true if they choose to use a random starting point, and returns false otherwise.

generateRandomStart function: Takes an integer that represents either the total rows or total columns on the board and generates a random number between 0 and the integer argument.

PrintAnt:

printAnt function: Takes the Ant object and a pointer to the Board object and prints the character representation of the board and ant.

Project 1 Design
Michael Nutt
CS162

Test Case:	Input Values	Function	Expected Output
Input too low	Input < 0	main(), After prompting user to start the program or for number of rows, number of columns, starting point, or the total number of steps.	Prompts user to enter valid input.
Input too high	Input > respective maximum value (e.g. Input > 100 for rows or columns)	main(), After prompting user to start the program or for number of rows, number of columns, starting point, or the total number of steps.	Prompts user to enter valid input.
Input is not an integer.	Input = 'a' Input = "a" Input = true; Input = 5.1; Input = 6a	main(), After prompting user to start the program or for number of rows, number of columns, starting point, or the total number of steps.	Prompts user to enter valid input.
Ant reaches an edge of the board.	rowCount = 10 colCount = 10 moveAnt(11, 0) moveAnt(10, 0) moveAnt(3, 11) moveAnt(3, 10) moveAnt(-1, 3) moveAnt(3, -1)	Board::isInBounds() Ant::reverseDirection()	The ant turns 180 degrees and moves one space in the new direction.
Menu function prompts user at end of program.	N/A	main(), while(programSelection == true)…	At the completion of the previous simulation, the user is prompted with the option to start the program again or quit.

Project 1 Design
Michael Nutt
CS162

Reflection:

Most of the changes I made to my original plan had to do with redundancies in how I was tracking the ant and the state of the board. For example, I had initially planned to store whether the user wanted to start the ant in a randomly-generated position within the ant object itself. Doing this meant I would have then had to also define another variable within the main function to store the same value, adding unnecessary complexity to the program. I also first tried to track the color of the square on which the ant was located as a property within the ant class which would in turn have to access the board object to get the color of that space. This created the same kind of chain of class property access that just added more complexity. By defining these variables within the main function, it was much easier to track and access their values with both the ant and board objects throughout the program.

I also had a bug where the space from which the ant just came would always switch to black ('#'), even if the space was black when the ant arrived there. After some debugging and logging, I realized that in my toggleSpace function, my first condition within the if/else statement was written "if (currentColor **=** ' ')". Thus, every time the toggleSpace function was called, that condition was considered to be true and the space would be set to black. Simply adding the extra "=" to create a comparison rather than an assignment fixed the issue.

When initially printing the board, I also had a bug where the program would only print the border, the ant, and any space the ant had already been to, but no others. It would just print the border with no whitespace and the '*' representing the ant. The problem occurred when the board was being initialized and printed, the print function was getting the current color of each space, but no color existed at that point. I had not accounted for that case in my getCurrentColor function, so it would just return an empty character as the value to be printed. To fix the issue, I updated my getCurrentColor function to default to whitespace if the space was neither black nor white.