Kimberly Tom
CS 225
4/11/18
Project 1 – Langton Ant

Design

1. Main
    a. Main is where we instantiate the board, moves, and menu.
        i. Key functions and data members
            1. hold pointers for location of ant
            2. collect number of user's move
            3. collect ant starting location
            4. collect board size
            5. display menu of actions
2. Ant
    a. Ant class will have the direction the ant is facing
    b. At white square, turn ant 90 degrees right and move forward one unit
    c. At black square, turn ant 90 degrees left and move forward one unit
    d. edge case: if ant hits the corner or side of the board, skip the step forward step, make ant turn 90 degrees in same direction, and have ant continue on
3. Board
    a. Board class generates the board
        i. generates the four sides of the board
        ii. change color of square the ant was on
        iii. represent black square with "#" character
        iv. represent white square with a space character
        v. represent ant with "*" character
4. Menu
    a. Menu class displays options to the user
        i. how many moves user wants
        ii. size of board user wants
        iii. starting position of the ant user wants
        iv. whether user wants to exit or start Langton's ant simulation
5. Input Validation
    a. check user's menu choice to tell user to choose again if invalid choice is selected
    b. if user enters anything besides 1 or 2, prompt menu again

Test Plan

TEST CASE	INPUT VALUES	DRIVER FUNCTIONS	EXPECTED OUTCOMES	OBSERVED OUTCOMES
input of board dimensions at negative	row < 0 col < 0	`checkBoardRows (int rows); checkBoardCols (int cols);`	invalid, ask user to input again	invalid, ask user to input again
input of board dimensions at zero	row 0 col 0	`checkBoardRows (int rows); checkBoardCols (int cols);`	invalid, ask user to input again	invalid, ask user to input again
input of board dimensions at 1	row 1 row 8, col 1	`checkBoardRows (int rows); checkBoardCols (int cols);`	invalid, ask user to input again	invalid, ask user to input again
input of board dimensions is positive	row 10, col 80 row 68, col 150 row 150, col 150	`checkBoardRows (int rows); checkBoardCols (int cols);`	valid, continue program to ant starting position	valid, continue program to ant starting position
input of board dimensions too large	row > 150, col > 150	`checkBoardRows (int rows); checkBoardCols (int cols);`	invalid, ask user to input again	invalid, ask user to input again
input of board dimension includes letter/symbol	row 5d col # row x99 col 5D	`checkBoardRows (int rows); checkBoardCols (int cols);`	invalid, ask user to input again	invalid, ask user to input again
input of ant starting position at negative position	row < 0, col < 0	`setAntInitial();`	invalid, ask user to input again	invalid, ask user to input again
input of ant starting position at zero	row 0 col 0	setAntInitial();	invalid, ask user to input again	invalid, ask user to input again
input of ant starting position at positive, within board bounds	row 5, col 44 row, 66, 6	setAntInitial();	valid, continue program to steps	valid, continue program to steps
input of ant starting position includes letter/symbol	row 77g col $ row x99 col c	setAntInitial();	invalid, ask user to input again	invalid, ask user to input again
input of ant starting position	ant row > board row	setAntInitial();	invalid, ask user to input again	invalid, ask user to input again

outside of board bounds	ant col > board col			
user selects to place ant randomly on board	selects random	`randomRow(int minRow, int maxRow randomCol(int minCol, int maxCol)`	places ant randomly within the bounds of the board	places ant randomly within the bounds of the board
step count negative	steps < 0	`checkStepCount (int steps);`	invalid, ask user to input again	invalid, ask user to input again
step count zero	steps = 0	`checkStepCount (int steps);`	invalid, ask user to input again	invalid, ask user to input again
step count positive	0 < steps <=20000	`checkStepCount (int steps);`	move on with simulation	move on with simulation
step count out of bounds	steps > 20000	`checkStepCount (int steps);`	invalid, ask user to input again	invalid, ask user to input again

all inputs within range	board: 13 x 55
	ant position 1, 55
	steps 12155
		`checkBoardRows (int rows);`
		`checkBoardCols (int cols);`
		
		setAntInitial();
		`checkStepCount (int steps);`
			shows the Langton ant highway, when ant his edge, wrap around board	shows the Langton ant highway, when ant his edge, wrap around board

all inputs within range	board: 150 x 2
	ant position 150, 1
	steps 4
		`checkBoardRows (int rows);`
		`checkBoardCols (int cols);`
		
		setAntInitial();
		`checkStepCount (int steps);`
			shows four steps of Langton ant	shows four steps of Langton ant

all inputs within range	board 150 x 150
	ant position 5, 99
	steps 20,000
		`checkBoardRows (int rows);`
		`checkBoardCols (int cols);`
		
		setAntInitial();
		`checkStepCount (int steps);`
			shows the Langton ant highway, when ant his edge, wrap around board	shows the Langton ant highway, when ant his edge, wrap around board

all inputs within range	board 2x2
	ant position 1, 2
	steps 250
		`checkBoardRows (int rows);`
		`checkBoardCols (int cols);`
		
		setAntInitial();
		`checkStepCount (int steps);`
			shows 250 steps of Langton ant	shows 250 steps of Langton ant

all inputs within range	board 6 x 7
	ant position: random
	steps 5,684
		`checkBoardRows (int rows);`
		`checkBoardCols (int cols);`
		setAntInitial();
		`randomRow(int minRow, int maxRow`
		`randomCol(int minCol, int maxCol)`
		
		`checkStepCount (int steps);`
			shows 5684 steps of Langton Ant	shows 5684 steps of Langton Ant

main menu user selects start program	1	`showMenu()`	starts program	starts program
main menu user selects terminate program	2	`showMenu()`	ends program	ends program
main menu, user selects start program, then start program again after program ran once	k 1, 1	`showMenu()`	runs simulation, then run simulation again, runs program, asks user again, then starts a new simulation	runs simulation, then run simulation again, runs program, asks user again, then starts a new simulation
main menu, user selects start program, then selects terminate program after program ran once	1, 2	`showMenu()`	runs simulation, then quits program	runs simulation, then quits program
Valgrind test for memory leaks	valgrind ./output	valgrind in vim	show no memory leaks	show no memory leaks

Project 1 Reflection

When I first thought about the design for the Langton Ant project, I did not initially think to do my input validation as a separate class. However, when thinking about it, I realized I would need input validation for many things, such as board, ant initial position, steps, and the main menu. I knew I wanted the menu input validation in the menu class though. However, I originally put my steps ant initial position input validation in the inputValidation class and it wasn't working. I then took it out of the inputValidation class and moved it to the ant class, because I needed to access the user specified board dimensions to check the ant initial row and column to make sure they are within the board dimensions, which I could not do in the inputValidation class.

Also, I originally had just a while and if statements for my input validation functions. I thought it was working because when I put in a letter, it wouldn't work. However, when I put in a number and a letter, it would use the number part as a valid input. I had to change my input validation to a do whole loop with string stream to prevent this from happening. It now correctly asks user to select again for combination of number/letter inputs.

Another problem I encountered was with the ant movement function. It took me a long time to figure out how best to go about it. I knew I wanted to use cases for each direction the ant was facing. I ended up coding each step in order. I wasn't quite sure how many if statements I needed for each case so I did this for around 12-14 steps. Then I figured out how many I would need for it to work and completed the remaining cases without testing each step. I eventually got it running correctly. Next, I had to figure out how to do edge cases. I figured wrapping around the board would be easiest because I would just have the ant be placed on either the minimum or maximum space for either the row or column. While I was coding this, I did run into some errors. This was due to not changing to the correct direction after the ant move. It took me a while to solve this one, but I eventually noticed that I had two

edge cases for north that weren't working and figured out it was something wrong with the north case and was able to fix it because I had it pointing to wrong direction after the ant move.

One last issue I had was I had a memory leak that took me a while to figure out. This memory leak actually lead me to find out that I had a big issue with my board function. While trying to solve the memory leak, I was fixing my initialization of boards and columns of the board. I found out that my code was making it so that I could not have a column count higher than the row count. Basically, I could only have a square matrix. I fixed my for loops when initializing the matrix. Regarding the memory leak, I called the makeBoard() function twice, once in the Board class and once in the Ant class. I didn't need to call it twice and this was also what was causing the memory leak. I got rid of the extra call to makeBoard() and ran valgrind, which then showed I had no more memory leaks.

I learned a lot from the Langton Ant project. At first, I was very overwhelmed as I didn't know where to start. I reminded myself that I needed to break it down into smaller parts, and work on things individually, then maybe the big picture will come to me. I changed my functions many times throughout the process, and moved some functions to different classes. I also learned it takes a lot of patience to get something correct. The cases for the ant moves were very time consuming. When I first worked on the ant move cases, I would do something for a while, then realize it doesn't work, and have to restart. Eventually, I got it to work, so the time it took for the trial and error paid off. This project also taught me that object oriented programming is very useful to keep things organized. It also taught me that code can be reused places with slight modification (for example, the input validation functions are very similar with a few adjustments). I do wonder if maybe there was a more systematic way to go through the ant movement cases, as I did each of them one by one and it was a little confusing which case I did and which case I didn't do yet. Overall, I think this project really made me think a lot about how different classes and functions interact with each other.