Design plan

Program name: **Langton's Ant**
Program purpose: Display an interactive simulation for the Langton algorithm.

1. **Main**
    - Purpose: game play
    - Functions:
        - Build board filled with white spaces
        - Move forward
            - If Ant is on white
                - Turn right 90 degrees
                - Turn space to black
            - If Ant is on black
                - Turn left 90 degrees
                - Turn space to white
        - Move forward until steps run out
        - Print board progression
    - Data members:
        - Board
        - Ant
    - Notes:
        - Ant: ^ char
        - White space: space char
        - Black space: # char
        - Moves: ~11,000
        - Board size : ~70x70
2. **Ant class**
    - Purpose: create and keep track of the ant
    - Functions:
        - Track ant's location
        - Track ant's orientation
    - Data members:
        - location
        - Orientation
3. **Board class**
    - Purpose: Track board state
    - Functions:
        - Keep track of white
        - Keep track of black
        - Keep track of ant (neither)
    - Data members:
        - Board
    - Notes:
        - Needs to know where ant is located
4. **Menu class/function**
    - Purpose: interactive menu
    - Functions:
        - Quit or restart when things are done
        - Prompt user for starting input
            - Verify input
                - Check type
                - Check not negative
                - Check ant position doesn't exceed number of rows/col
                - Doesn't crash on undesired input
            - Get number of rows
            - Get number of columns
            - Get number of steps
            - Starting row of ant
            - Starting column of ant
        - EXTRA CREDIT: option to start ant at random spot
            - Tell user at the beginning about EC
    - Data members:
        - Input
        - rows
        - columns
        - steps
        - Ant's starting position
    - Notes:
        - Should be easily changeable
5. Edge Case
    1. Hitting the edge of the board
        1. Just turn again and apply same black/white logic

Test plan

#	Test	Input	Targeted Function	Expected Output	Actual Output
			INPUT VALIDATION		
1	Row must be >= 70 and <= 100	1. 69 2. 101 3. 70 4. 100	menu.getInfo menu.minCheck menu.maxCheck	1. & 2. Error asking for value in range 3. & 4. Column (next) prompt	1. & 2. Error asking for value in range 3. & 4. Column (next) prompt
2	Column must be >= 70 and <= 100	1. 69 2. 101 3. 70 4. 100	menu.getInfo menu.minCheck menu.maxCheck	1. & 2. Error asking for value in range 3. & 4. Ant starting position prompt	1. & 2. Error asking for value in range 3. & 4. Ant starting position prompt
3	Steps count doesn't exceed 20000	1. 20001 2. 20000 3. 12000	menu.getInfo menu.maxCheck	1. Error asking for value in range 2. Regular game play 3. Regular game play	1. Error asking for value in range 2. Regular game play 3. Regular game play
4	Steps count is above 0	1. -1 2. 0 3. 1	menu.getInfo menu.minCheck	1. Error asking for value in range 2. Error asking for value in range 3. Regular game play	1. Error asking for value in range 2. Error asking for value in range 3. Regular game play
5	Ant starting position doesn't exceed bounds of board (rows & columns)	Rows: 70, Columns: 70 1. 0, 90 2. 90, 0 3. 80, 85 4. 0, 70 5. -1, 65	menu.getInfo menu.minCheck menu.maxCheck	1-5. Error asking for value in range	1-5. Error asking for value in range
6	Each prompt rejects char (6 prompts)	1. hgjhjf	menu.intCheck	1. Error asking for integer	1. Error asking for integer
7	Each prompt rejects floats	1. 1.11	menu.intCheck	1. Error asking for integer	1. Error asking for integer
			ANT MOVEMENT		
8	Turns to next valid location when hitting board bounds:	1. Place ant in top right 2. Place ant in top left 3. Place ant in bottom right 4. Place ant in bottom left	ant.play	1.- 4. While staying in the same spot, the ant rotates the appropriate direction (black - left, white - right). When it can move forward in bounds, it does so and regular game play continues	1.- 4. While staying in the same spot, the ant rotates the appropriate direction (black - left, white - right). When it can move forward in bounds, it does so and regular game play continues
9	When on white, ant turns 90 degrees right and changes space to black	1. Iteration of play function where ant is on 'white' space	ant.play	1. Orientation turns "right" 2. Ant moved 3. Last space ant was on is black	1. Orientation turns "right" 2. Ant moved 3. Last space ant was on is black
10	When on black, ant turns 90 degrees left and changes space to black	1. Iteration of play function where ant is on 'black' space	ant.play	1. Orientation turns "left" 2. Ant moved 3. Last space ant was on is white	1. Orientation turns "left" 2. Ant moved 3. Last space ant was on is white
			OTHER FUNCTIONALITY		
11	Play again keeps going with 1	1. 1	menu.getInfo	1. Prompts restart	1. Prompts restart
12	Play again quite game with 0	1. 0	menu.getInfo	1. Good game message, program ends	1. Good game message, program ends
13	Prints board properly (rows/columns)	1. Finish prompts	ant.makeBoard ant.print	Print out board with: 1. Correct rows 2. Correct columns 3. Ant in correct position	Print out board with: 1. Correct rows 2. Correct columns 3. Ant in correct position

Reflection:

Design Implementation:

In the future, I think I will spend more time on programming and really figuring out what exactly the class was going to do. I ended up with a much more extensive menu function than I intended. I also intended to have a Board class, but as I coded, I realized I didn't really need a class to just print the board. Plus, having the board stored within the Ant class helped limit some complexity since I didn't have to pass extra parameters by reference or pointer.

I turning of the ant was particularly challenging. I ended up going back to paper and creating diagrams by hand for a few edge encounters. These visuals really helped me figure out how the ant was supposed to move and when the movement was actually buggy.

Time management was another big key with this project. I think I underestimated not only the time for this individual assignment, but for all my other assignments as well. In the future, I want to manage my time better with projects and balancing other classes.

Overall I think next time I'll be putting in more time for everything as early as possible. Furthermore, I will really focus on planning. My style of planning for this project would've been good enough for labs (which are simpler), but something more thoughtful would be more helpful. I think I deviated much more than I expected from my plan. Next time, I'd like to close the gap between planning and practice.

Test Plan:

I found the test plan extremely helpful when figuring out whether my program was 'done'. I could make clear tests based on the components of the rubric, allowing me to 'remember' the small details that often get left out. I think the 'Targeted Function' column helped me see where I wasn't testing my program as well as areas I may have been over testing.

I think the test plan in particular helped me figure out the design better. Some times with the design portion it's easy to get overwhelmed with what to do next. However, the testing plan makes you focus on the small things you need to test. This helps reveal gaps in the design and where I should put more thought into how I was going to do something.