Robert Elsom

Project 1

Design

1. Ant class
    a. Contains the getters and setters pertaining to the ant. Responsible for both location and orientation.
    b. List of key functions
        i. setLocation
        ii. getLocation
        iii. setOrientation
        iv. getOrientation
2. Menu function
    a. Responsible for starting menu along with starting positions and board size. Also contains the options to quit before and after the simulation is run.
    b. List of key functions and data members
        i. Ask user to start simulation or quit
        ii. If starting simulation, ask user for number of rows and columns for the size of the board, number of steps during the simulation, and either a random starting position or starting row and column of the ant.
        iii. Once simulation is over, ask if wants to restart or quit
        iv. If quit is chosen, exits program
3. Input Validation Function
    a. Responsible for making sure that input is valid to make starting prompt and menus work.
4. Board class
    a. Responsible for outputting the board and tracking and changing the square colors. Must keep track of the color of the squares including the square that the ant is currently occupying.
    b. List of key functions and data members
        i. makeBoard
            1. creates dynamic 2d array
        ii. drawBoard
            1. outputs the board to the screen
        iii. changeColor
            1. changes color of the square
        iv. antSquareColor
            1. contains the color of the square the ant is in until the ant moves, then sets the color of the square to the color the ant changed it to

Test Cases

Starting Menu test cases

Test case	Input Values	Driver Function	Expected Outcome	Observed Outcome
Function a not an unsiged integer	A, 1.5, -4	ValidStr()	Display error, repeat options to user	Display error, repeat options to user
Integer not a valid option	5	ValidStr()	Display error, repeat options to user	Display error, repeat options to user
Integer a valid option	1	Menu() If choice == 1 || choice ==2	Print starting prompts and get inputs for functions	Print prompts and get inputs for functions
Integer second option	2	Exit()	Quit program	Quit program

Repeat Menu Test Cases

Function a not an unsiged integer	A, 1.5, -4	ValidStr()	Display error, repeat options to user	Display error, repeat options to user
Integer not a valid option	5	ValidStr()	Display error, repeat options to user	Display error, repeat options to user
Integer a valid option	1	Menu() If choice == 1 || choice ==2	Print repeating prompts and get inputs for functions	Print prompts and get inputs for functions
Integer second option	2	Exit()	Quit program	Quit program

Board Row and Column Input

Function a not an unsiged integer	A, 1.5, -4	ValidInt()	Display error, repeat options to user	Display error, repeat options to user
Integer not in range (<1 or >400)	-5, 0, 500, 1000	ValidInt()	Display error, repeat options to user	Display error, repeat options to user
Integer in valid range (1 = 400)	1, 444, 15	Menu()	Store in variable and in response array	Store in variable and in response array
Input at extreme low	1	Menu()	Store in variable and in response array	Store in variable and in response array
Input at extreme high	400	Menu()	Store in variable and in response array	Store in variable and in response array

Number of Steps Input

Function a not an unsiged integer	A, 1.5, -4	ValidInt()	Display error, repeat options to user	Display error, repeat options to user
Integer not in range (<0 or >10000)	-5, 0, 10001, 55555	ValidInt()	Display error, repeat options to user	Display error, repeat options to user
Integer in valid range (1 = 400)	1, 444, 15	Menu()	Store in variable and in response array	Store in variable and in response array
Input at extreme low	0	Menu()	Store in variable and in response array	Store in variable and in response array
Input at extreme high	100000	Menu()	Store in variable and in response array	Store in variable and in response array

Player Starting Positions Input

*note, each player input has corresponding board variable. playerRow corresponds to boardRow, playerCol corresponds to boardCol.

Function a not an unsiged integer	A, 1.5, -4	ValidInt()	Display error, repeat options to user	Display error, repeat options to user
Integer not in range (<0 or > board variable - 1)	-5, 500, board variable	ValidInt()	Display error, repeat options to user	Display error, repeat options to user
Integer in valid range (0 to board variable - 1)	0, 15, 23 (given board variable is bigger than 15 or 23 on test)	Menu()	Store in variable and in response array	Store in variable and in response array
Input at extreme low	0	Menu()	Store in variable and in response array	Store in variable and in response array
Input at extreme high	Board variable - 1	Menu()	Store in variable and in response array	Store in variable and in response array

Reflection

In project 1, I had to create my main function larger than originally planned. After restarting the entire project, I realized that creating the array in the main function instead of making it in the board class and having a get function to return the address. I kept trying to pass the address back and use it, but when writing other parts of the function was making mistakes and creating segmentation faults.

The other problem I ran into that I did not realize until redesigning my program was that I was declaring multiple Board objects in different functions and classes and giving them the same name. This meant that I was having a lot of trouble keeping things straight and was thinking the object should be the same between classes. It was not until later, after moving my array representing the board to the main function that I realized that each object was different and this was the main reason I kept getting segmentation faults. To correct my problems, I created a larger main function and decided to make the actual moves step by step in the function instead of trying to create one function to make the moves.

Another problem I ran into was trying to make my menu function reusable. I decided to validate the input in separate functions, validStr and validInt, that return the input if it passes the test instead of using a lot of for and while loops inside the menu function. Another change that I made was when printing the menu, I added a parameter call in order to go from the starting prompts to repeat prompts. This allows me to keep the same function, and whenever I want call a repeat prompt, I pass a bool at the end of the call. The last change I had to make to my menu function was store all the prompts and titles in strings at the beginning of the function. This allows reuse easier in that I only have the change the strings at the beginning instead of searching throughout the whole program to change every string. Another change that I made from the original design is to make the menu function an int, to return an array of integers containing user input for board size, starting position, and number of steps. In the future, if I needing to return something other than an int, changing the menu from an int to a string function and returning everything back in a string. It was not until writing this reflection that I realized that creating it this was first is probably the better option.

This program was definitely challenging. Creating everything with just a few parameters was a change of pace from the last semester, and something that I grew to enjoy by the time that I finished. This was the first time it took me more than one sitting to finish an assignment, and was also the first time I had to abandon my original plan and go back to redesigning the entire program. On redesign, I realized I was making things unnecessarily complicated, and something I will remember next time I am designing a program.