Ahmad El-Bobou

Fall 2018

162

Project 1

Program Design

For this project, we are instructed to create a Langton's Ant simulator. The requirements are the the following:

1. We follow the Langton's Ant Rule for movement.
2. We must use an Ant Class.
3. The board must reflect the ant's movements and display its current location.
    a. Remember edge cases
4. We must use menu functions and all input should be validated.

The first problem is how to create a board that is dynamically allocated. The only solution I am aware of is to make a dynamically allocated 2d array. This array will exist in the main function and will be deleted before the program terminates.

At the beginning of the program, the user will be prompted if they want to play of quit, and if they choose to play, they will have to input the dimensions of the board and how many turns they desire the ant to move. This information will be stored in variables in the main.cpp file and will be fed to the Ant class by passing the variables and the array as references. After the initial menu asking if the users wants the play, the entirety

of the code will be housed in a while loop that will function so long as the user has not quit.

Building the board will also take place by the ant class. A function will create/reset the whole board to white. This function's parameters will be the size of the board and the dynamically allocated 2d array. There will be a separate function to run the game. In this function, the ant's location, movement, and orientation will be pulled from private attributes of the ant class. The default location of the ant will be the manually input or selected to be random by the user. In this method, there will be a for loop that will iterate as many times as the user wants the ant to move. Inside this for loop, there will be a series of if statements that check the ants orientation, location in order to determine the color of the tile it is on and which tile it will go to. Once the ant moves, there must be a way to change the previous tile to the correct color and the current tile to and "*". Within this function another method from the Ant class will be called to display the board. As of right now, I plan to print the board repeatedly so the user can review the movements of the ant.

There must be code housed within the function to run the game which controls the movement of the ant. We must keep track of the x/y coordinates in order to manipulate the ant. If the x/y coordinates go out of bounds, they will wrap the ant around the board by setting them to the other extreme.The x/y coordinates will be set equal to the starting location. Then, depending on the orientation, the ant will move. The orientation will consist of 4 directions (UP, RIGHT, DOWN, LEFT) depicted by the integers 1-4 and enums. If the orientation goes smaller than 1 or larger that 4, the

orientation with be set to 4 or 1 respectively. In order to keep track of the color of the tile on which the ant is on, a secondary bool variable will be used to store the color of the ants location in that iteration of the code. This is necessary because after the ant moves, the tile must be set to a color.

Once this function has been called and the ant simulation has ended, the program will run another menu function that will ask the user to play again.

Test Plan

Test Case	Input	Expected Output	Actual Output
Does program quit?	1. n	1. Program quits	1. Program quits
Is initial function case sensitive?	1. N	1. Program quits	1. Program quits
Does the program run well with appropriate input?	1. Y 2. 25 3. 5 4. 10 5. N 6. 5 7. 1 8. n	1. Menu prompts 2. 25 columns 3. 5 rows 4. Ant moves 10 times 5. Manually enter starting position 6. Ant starts on row 5 7. Ant starts on column 1 8. Program ends	1. Menu prompts 2. 25 columns 3. 5 rows 4. Ant moves 10 times 5. Manually enter starting position 6. Ant starts on row 5 7. Ant starts on row 1 8. Program ends
Are negative	1. Y	1. Menu	1. Menu

numbers accepted as input?	2. -10 3. 10 4. -5 5. 5 6. -20 7. 20 8. P 9. Y 10. n	prompts user 2. Error - enter int over 0 3. 10 columns 4. Error - enter int over 0 5. 5 rows 6. Error - enter int over 0 7. Ant moves 20 times 8. Error - enter character y or n 9. Random starting location 10. Program ends	prompts user 2. Error - enter int over 0 3. 10 columns 4. Error - enter int over 0 5. 5 rows 6. Error - enter int over 0 7. Ant moves 20 times 8. Error - enter character y or n 9. Random starting location 10. Program ends
Are non-integers accepted as inputs? Does the program restart?	1. y 2. t5 3. 10 4. 00d5 5. 5 6. 20. 7. 20 8. P 9. y 10. Y	1. Menu prompts user 2. Error - enter int over 0 3. 10 columns 4. Error - enter int over 0 5. 5 rows 6. Error - enter int over 0 7. Ant moves 20 times 8. Error - enter character y or n 9. Random starting location 10. Program restarts	1. Menu prompts user 2. Error - enter int over 0 3. 10 columns 4. Error - enter int over 0 5. 5 rows 6. Error - enter int over 0 7. Ant moves 20 times 8. Error - enter character y or n 9. Random starting location 10. Program restarts
Does the program repeat well when input	1. Y 2. 25 3. 5 4. 10	1. Menu prompts 2. 25 columns 3. 5 rows	1. Menu prompts 2. 25 columns 3. 5 rows

is entered correctly?	5. N 6. 5 7. 1 8. y 9. 50 10. 20 11. 100 12. y 13. n	4. Ant moves 10 times 5. Manually enter starting position 6. Ant starts on row 5 7. Ant starts on column 1 8. Program restarts 9. 50 columns 10. 20 rows 11. Ant moves 100 times 12. Random starting location 13. Program ends	4. Ant moves 10 times 5. Manually enter starting position 6. Ant starts on row 5 7. Ant starts on column 1 8. Program restarts 9. 50 columns 10. 20 rows 11. Ant moves 100 times 12. Random starting location 13. Program ends
Does the pattern for the ant emerge with sufficient iterations and board space?	1. Y 2. 100 3. 100 4. 11000 5. Y 6. n	1. Menu prompts 2. 100 columns 3. 100 rows 4. Pattern emerges with this many steps 5. Random starting location 6. Program ends	1. Menu prompts 2. 100 columns 3. 100 rows 4. Pattern emerges with this many steps 5. Random starting location 6. Program ends
Does the program allow zero to be taken as an input?	1. y 2. 0 3. 10 4. 0 5. 5 6. 0 7. 20 8. N 9. 0 10. 5	1. Menu prompts 2. Error 3. 10 columns 4. Error 5. 10 rows 6. Error 7. 20 moves 8. Manual starting	1. Menu prompts 2. Error 3. 10 columns 4. Error 5. 10 rows 6. Error 7. 20 moves 8. Manual starting

	11. 0 12. 3 13. n	location entry 9. Error 10. Ant placed on row 5 11. Error 12. Ant placed on column 3 13. Program ends	location entry 9. Error 10. Ant placed on row 5 11. Error 12. Ant placed on column 3 13. Program ends

Reflection

After many hours and errors I have completed this monster of a program. I have to say that this was definitely challenging. This was the first time I took words to paper and then those words to pseudocode before typing one thing in vim. I definitely got more comfortable with vim through this project and feel definitely more competent than before. In this project, I solidified my knowledge of dynamically allocated 2d arrays and classes. I also got a good review on diverse loops when practicing input validation.

In terms of what I changed, I didn't stray from my original design too much. In my initial design plans, I didn't explicitly state how input validation would occur. I ended up creating a separate hpp and cpp folder to house those functions. The menu and input validation functions should definitely be useful for later projects. I also opted to complete the extra credit and made an option for the user to have the starting position be random.

Some problems I encountered included issues using the srand() and rand() functions. It had been quite a while since I used them and I needed to find a safe place to put the initial srand() call so the numbers generated for the ant's starting position would actually be random for the duration of the program. I ended up putting it in the main function and that seemed to

work just fine. Another issue I had was that I initially got the x and y axes mixed up on the board. This was causing the board to display at a 90 degree angle. After I resolved this, the board displayed as it should. Another issue I encountered was when I attempted to use flip and have the program iterate over 10,000 times. The delay was too much, so I ended up only printing the board once in the instances that I tested the high number of iterations.

Overall, I really enjoyed this project and learned a ton working through it. I look forward for what is to come.