Ethan Dunham

dunhamet@oregonstate.edu

Project 1

Design and Reflection

1/22/17

Design:

Problem to be solved:

I need to Make an ant that follows predesignated directions for certain requirements while within a matrix. At a white spot, turn right, change the color of the spot, and move forward one spot. At a black spot, turn left, change the color of the spot, and move forward one spot. Create a way to randomize the start location of the ant and the direction it is facing. Create a menu driven system to give the user options for running the program. Work out validation to make sure the ant is still in the matrix. Make a board class for the current board to keep track of current and the past few moves to determine which direction the ant is facing. Create a function in board that displays the current board. Put a short pause between displays so that the user can watch the ant move.

Main

Displays a menu.

1. Diameter
2. starting x
3. starting y
4. direction ant is facing
5. run

Takes the option picked and calls that menu function from the menu class, then puts its return value into the corresponding variable.

Maybe have 0 designate random numbers.

Ant class

Ant

Constructor with current spot of ant, direction it is facing, number of moves, and size of board.

setdirection

Keeps track of ant facing direction each turn by remembering past values and adjusting accordingly.

Make move

Calls board with the coordinates of the ant and then gets the color of the board to determine the move. Then calls the correct color move by using move"color."

validMove

checks the current move to verify if it was valid based on if the ant went out of bounds, etc.

Board Class

Board

Writes an old board and current board. Old board will be written before ant so that color of ant spot can be remembered.

moveBlack

what ant does on a black spot

turn left, change the color of the spot, and move forward one spot

have an if statement for each facing variables with what to do.

Depending on the facing direction, the X or Y coords of ant will have to be incremented or lowered by 1.

Up= X-1

Down=X+1

Left = Y+1

Right=Y-1

Change direction ant is facing.

moveWhite

what ant does on a white spot.

turn right, change the color of the spot, and move forward one spot

have an if statement for each facing variables with what to do.

Depending on the facing direction, the X or Y coords of ant will have to be incremented or lowered by 1.

Up=X+1

Down=X-1

Left = Y-1

Right = y+1

Change direction ant is facing.

printBoard

prints the current board.

getX

sends x coords

getY

sends y coords

Verification

Verification function that checks values for greater than, less than, etc.

Menu class

Menu class that is called from main to fill in the values of each option.

Asks for diameter, starting X and Y, and direction the ant is facing.

Validate each entry to make sure it works.

Make sure all numbers are greater than 0 and less than 1000?*

*check size of putty on full screen later to determine max.

Reflection:

The logic of the ant's motions was easy to determine. The validation of everything so that it would run smoothly was more difficult.

Change log:

I was originally going to use a board class and an ant class to run the program. However, I had a lot of issues calling the board class from within the ant class. I tried various #include combinations, searched online, etc. but could not get, for example, board1.writeOldBoard() to work. It constantly gave me issues about undeclared variable. After this conundrum, I decided to, against my original judgement, to consolidate both classes into a single ant class. This ended up being a great decision. By doing this, I was able to remove issues with accessing different variables, constantly sending x and y coordinates back and forth, and was able to clean up the code by reducing the number of member functions. Since each function could already access the variables, I was able to remove getX, getY, setColor, as well as several other functions.

Originally, I terminated the ant when it got to the edge of the matrix. After checking Piazza, it was determined that was not the way to do it, though it was later reversed. After trying the first 2 options listed at the time, I noticed the ant would just get stuck or it would mess up the greater scheme of the matrix, so I implemented a wrap around when it reaches the edge. Since this validation was completed in the moving functions, I no longer needed a validMove function to verify the move. Thus, I was able to clear up more space and trim down the code a bit more.

The menu class became a menu function in the final product. It became a pass by reference to the values in main. This will make it easier to use in other projects because the only thing needed to be changed is the number of cases and what is on each switch case. I also changed the options. After reading into the project more, I discovered that it was not stated that it would be a square matrix, so the diameter option had to be split into rows and columns. I also added an option for number of turns for the user to pick instead of being random. I did keep the randomly generating numbers if they are still value 0, that way if the user wants to switch back from an inputted number back to random, they can do so.

The user validation functions, which I misunderstood from the project description, was used sparingly until I learned from Piazza that it should be used to stop cin issues. So after using Google to figure out how to go about stopping cin issues, I discovered cin.fail(). Cin.fail() became quite useful in my program. For every cin statement, I created a do while loop which checked cinFail(). If it failed, the cinFail function I created would clear and then ignore 300 spots of cin before looping back to the question.

One of the biggest issue I had was fixing every conceivable way to break the program. Cin.fail was very useful in this. I also had to do some tricky validations with the random numbers that could be generated. If the user picks a random start location, but not the size of the matrix, I had to account for that, as well as how to validate if they chose a start location within their current sized matrix, but then went back and only changed the size of the matrix and not the starting location. After many hours of breaking my program, I believe I solved most or all issues that break it.