COMPUTER SCIENCE II
ASSIGNMENT 1 REFLECTION
TUCKER DANE WALKER

PROGRAM DESIGN

DESIGN DESCRIPTION

Problem to be Solved

1. I need to create a program that prints out an image of a 2D “sandbox” which contains an ant moving around in an ordered fashion inside of the box
2. The ant must be a separate entity from the sandbox
3. The 2D sandbox must have two types of tiles or spaces. These are “black” and “white”
4. The 2D sandbox must be iteratively updated after each output based on movement of the ant
5. The ant must also be displayed in the printed image of the sandbox
6. The ant must be able to start in any place within the bounds of the sandbox with a user-defined direction it is facing (a cardinal direction), and a grid coordinate within the box
7. The ant must “walk” for a user-defined number of steps before the program is complete
8. The starting place of the ant must also have the option of being randomly generated
9. There needs to be some sort of menu which validates user input and controls what the user can/cannot input for the ant to do; this input validation function needs to be versatile so that it can be re-used.
10. The ant must move forward in the cardinal direction of travel between each update of the screen and then “turn” and change its cardinal direction based upon the color of the square it moves into
11. When the ant hits a white square it must turn 90 degrees to the right, when it hits a black square it must turn 90 degrees to the left
12. The ant must also “flip” or change the color of the square it lands on after turning

Inputs/Outputs

1. User Input needs to be as follows:
    a. Size of sandbox or 2d Matrix, both x and y axes
    b. Number of steps for the ant to take
    c. Starting location of the ant
2. Sandbox or Matrix will be a class whose objects take input and provide output using get() functions. These should be something like:
    a. Get the actual value of the square at x,y (black or white)
    b. Get what should be printed at square x,y (black, white, or the ant)

    c. Get the size of the matrix (returns the size)
    d. Needs to have a constructor/destructor, especially if there is a dynamically allocated 2d matrix involved
3. Ant must also be its own class
    a. The ant should record its own position, so one should request that position with a get function and have its position as output
    b. The ant should also have a cardinal direction that can be gotten as output
    c. The ant should be able to take or sense the color of the square it sits on and turn based upon the color of the square. The turn can be considered the result or output from the “input” of color
4. There also needs to be a menu
    a. The menu must take arguments for choices given by the user
    b. The menu must validate these arguments
    c. The menu must then throw an error if the argument does not fit within the correct range of possibilities or return the output for the choice/choices to be used.

Algorithm Pseudocode

Main Function:

1. Create Matrix
2. Create an ant to live inside of the matrix or sandbox
3. Request a number of steps from the user using a menu
4. Run the ant simulation within the matrix for the desired number of steps

Ant

1. The ant must have the following variables
    a. Xposition
    b. Yposition
    c. CardinalDirection
2. The Ant must have the following functions
    a. setXposition getXposition get/set an integer
    b. setYposition getYposition get/set an integer
    c. setCardinalDirection getCardinalDirection
        i. get/set an integer associated with a direction (up, down, left, right = 1,2,3,4)
    d. moveForward
        i. Changes the value of Xpos and Ypos based on the CardinalDirection
    e. Turn
        i. Changes the Cardinal Direction of the ant

Matrix

1. The Matrix must have the following variables
    a. Size (x and y axis lengths)
    b. Values for black and white (bool?)
    c. A dynamically allocated matrix containing said values

2. The Matrix must have the following functions
    f. getSize setSize get/set x/y axis lengths
    g. setPositionValue sets a position to black/white
    h. printMatrix print what is in the Matrix at a position
3. The Menu must have the following functions
    a. Choices a programmer defined number of choices with associated values
    b. A validation a way to validate that the programmer-defined choices are forced

Test Plan

I tested my output by adding tests within Main that output information like “the ant is facing the cDirection direction”, “the ant is turning now”, “The ant is at square x, y”, “The square is (black or white)” etc. This method of testing allowed me to see the individual steps that were happening and compare them to what I saw as output on the matrix. This not holy helped with my “turning” problem, but allowed me to ensure the correct order and sequence of the ant’s movement: Move, then Turn, then flip the color of the board, then another move. Much of my test output is included as comments in the Main.cpp file.

I also tested each function of my code individually by providing printed output to console and compiling/running that section of code using dummy input. Not all of these tests are provided in my final draft of the assignment because that would be extraneous. These tests were exhaustive.

Test Results, Problems and their Resolution

After working on this assignment, I identified a number of problems that I needed to solve in order to ensure that I had the code working effectively. One problem that I encountered was that the ant might hid the edge of the matrix. I solved this by allowing the ant to wrap around.

Another problem that I had when trying to validate and test the results of my code was that the output was difficult to discern using “#” for black and “ “ for white. I chose to switch these which made this much easier because I could see the actual bounds of the matrix when output to the screen.

It took some time for me to work out the “turn” function for the ant and get it to work by setting the cardinal direction properly. Initially I had my cardinal directions: 1=up, 2=right, 3=down, 4=left rotated to the right one in my code, so the ant would not be facing the right way. Because I had an output of what was happening during each step, I was able to quickly identify this problem and fix it.

Initially, I was thinking that I needed to make the ant an object that was a part of the matrix. This is a possibility, though I found it easier to make them two separate and distinct entities. The way

I thought about it was that a sandbox can have an ant on it, but the ant is not a part of the sandbox per-se. This allowed me to change the values of the sandbox tiles independently of the position of the ant. What this means then, is that the value of the sandbox at [1][1] could be “black” and the ant could still be sitting on [1][1] without it being set to “ant”. This way, if I needed to use the matrix again in the future, I would be able to request the value at [1][1] and still get the correct color of the board. I was able to solve printing the ant on top of (or instead of) the color using a separate print function in matrix. The print function checked to see if the ant was sitting on [x][y]. If it were, it would output an “*”, otherwise it would print “#” or “ “ for white and black.

For input validation, I chose to create a menu that requires 4 options that the programmer wants to give to the user. This can be expanded in the future. The format of the menu is:

> “Enter the “ (values) “ for the “ (entity).
> User enters an integer associated with one of 4 values defined by the programmer.

The program then checks to see if the integers provided by the user are 1, 2, 3, or 4. If they are not, then it throws an error and continues to request valid input until it is given.

The menu then returns an integer for whatever choice was given which can then be used in the rest of the program.