Cody Hannan
CS162

PROJECT 1 - REFLECTIONS

DESIGN DESCRIPTION

The project we are tasked with is to create a simulation of Langton's Ant. The rules for the ant include turning right after entering a white space and turning left after entering a black space. When it leaves the space it changes it to the opposing color. We also need to show the ant's location as it moves about the board.

A way to go about this, is by creating an Ant class that contains functions for initializing the board, displaying the board, placing the ant on the board, and another function that changes the state of cells as the ant leaves them. We would also want a function that takes user entered values for the rows and columns of the board, the coordinates for placing the ant, as well as its starting direction, number of steps, and its speed. This function would then contain the behaviors for the ant and would utilize the Ant class functions to essentially begin its movement and display it to the user.

The main function would mostly act as a way to ask the user for the different parameters, which would contain a menu for updating the parameters, and would also include methods for input validation in case the user were to enter an unexpected value.

Pseudocode:

Basic menu

```
While(choice!=1 or 2 or 3 or 4)
{
Initialize variables

Cout messages that show the user what options there are like:
1.Quit
2.Change Board dimensions
3. Change Ant starting location and direction
4. Run program
Cin >>choice

    Switch (choice)
    {
        Case 1:
        {
            Uses break operator to leave switch and end program
        }

        Case 2:
        {
            Initializes relevant variables back to something incorrect so that input validation
            is triggered
```

Asks user to input values for brow and bcol

sets variable choice to 0

Case 3:

{

Initializes relevant variables back to something incorrect so that input validation is triggered

Asks user to input values for arow and acol and to enter either n, s, e, or w for direction

sets variable choice to 0.

}

Case 4:

{

Calls playAnt function that uses user entered parameters to begin ant movement and display it to the user

sets variable choice to 0.

}

}

}

Ant class

Private:

variables will be ints for rows and columns, and a double pointer to a 2d array called boardArray

Public:

Ant(int x,int y) -- constructs the array by allocating space in memory for the array and then initializing each space to “white”

~Ant() -- deallocates space in memory to avoid leaks

Void printBoard(int speed) -- displays the array to the user

Char setAntBoard(int arow, int acol) -- stores state of cell in some variable - places ant - returns the stored state

Void antTrail(int prow, int pcol) -- changes cell states based on where the ant has been

playAnt function

Creates Ant class object a1(brow,bcol), which initializes the field.
For loop that repeats based on the value for steps
{

Uses user entered parameters for arow, acol, and direction with the Ant class function setAntBoard to place the ant within the array. The function setAntBoard returns the state of the cell at arow and acol before the ant is placed.

If the state returned is “white”
{

switch(dir)
{

Case 1: north - changes dir to east - acol++
Case 2: south - changes dir to west - acol--
Case 3: east - changes dir to south - arow++
Case 4: west - changes dir to north - arow--

}

Else if the state is “black”
{

switch(dir)
{

Case 1: north - changes dir to west - acol--
Case 2: south - changes dir to east - acol++
Case 3: east - changes dir to north - arow--
Case 4: west - changes dir to south - arow++

}

}

Print function from Ant class is called to display board

antTrail function for Ant class is called to change state of cell where ant is currently placed to black or white depending on if the space previously held a white or black state respectively

}

Input Validation Template

Initialize variables to something outside of where they should be
while(variables are outside of where they show be or entered value isn’t of the correct type)
{

Cout << asks user for some parameter
Cin >> user enters something

if(entered value isn’t of the correct type)
{

```
            Clears whatever was entered
            Cout << asks user to enter something of the correct type
        }

        if(entered value is outside of the correct range)
        {
            Cout << asks user to enter something within the correct range
        }
}
```

TEST PLAN / RESULTS

Test Case	Input Values	Expected Outcomes	Observed Outcomes
Small Array	brow=1 brow=1	Tells user to enter values that are at least 2	Tells user to enter values that are at least 2.
Ant outside of field	arow<0 acol<0 arow>brow acol>bcol	Tells user to enter value that is at least zero and less than field dimensions	Tells user to enter value within proper range of the field
Ant in a corner of the field	arow=0 acol=0 arow=brow acol=bcol	Depending on which direction ant is facing it should turn around and move in the opposing direction	Receive an error if arow=brow acol=bcol
Randomly place ant	Select option to randomly place ant	Randomly places ant within field	Randomly place ant within field
Ant bordering each wall of the field	arow=0 acol=0 arow=brow acol=bcol	If ants direction would move it outside of array then it should turn directly around	Receive an error if ant hits the south and east walls. With north and west it works properly
Inputs of improper type	Double Char String	Tells user to enter correct type and loops back to ask for an input	Tells user to enter correct type and loops back to ask for an input
Inputs of improper range	Negative number Number below range Number above range	Tells user to enter number that is within correct range	Having issues with coordinates for ant compared to field dimensions

Menu option “Run Program” is selected before all parameters are entered	Select run program from menu	Tells user to finish entering all parameters beforehand	Tells user to finish entering all parameters beforehand
Menu allows updating of individual parameters	Choose menu option to update after having run program	Allows user to update parameters	Allows user to update parameters

REFLECTIONS

The main issue I ran into was how to go about organization. I had a hard time figuring which functions should belong to what class or how many classes I would need. Another issue was storing values in the 2d array across multiple files. I found that allocating memory space by using the new operator and a double pointer dealt with the issues, though I was reluctant because I still have problems understanding the syntax for pointers. The other problem areas were input validation and setting up the menu. I debated as to whether or not I should have found a way to make these as either separate functions or classes, but ended up not having enough time to try to figure that out. The solution I came up with (throw it all in the main) worked but wasn’t as clean as I would have liked. This project had a lot of moving parts outside of just have the ant behave properly. For me, the complexity came from implementing the menu system and input validation.

Another issue I ran into was making sure the ant behaved properly. In particular with how it reacted when it came to the edge of the array. This issue was brought up on Piazza and one of the okayed solutions was to turn the ant around, which is what I decided to go with. One error that I hadn’t noticed right away was that when the board was initialized the array dimensions would be from 0 to 29, for instance, rather than 1 to 30, which became a problem when setting the ant position near a wall and using a comparison between the ant coordinates and the dimensions of the array. The intended placement at (15,15) would actually need to be (14,14). This affected a number of comparisons that were used to determine behavior as well as input validation. My solution has been to not let the user place the ant next to the board boundary and instead have at least a space in between.

I’ve noticed that my typical “plan” is to dive in and slowly figure out an organization that works, but may not be the best and also has the added benefit of making the development process much more difficult. At the moment my project planning and code planning skills are something I really need to work on. I think over time with experience programming more projects I will improve as I begin to internalize the ways in which I solve these problems.