Lab 1 Design + Reflection

Design

Menu file {

startMenu function {

display option to start game or quit

input validation via while statement

}

getValues function {

retrieves number of rows, number of columns, starting row, starting column and steps all received by reference and passed back to main

use while function to run menu until option to start program is selected

use while function for input validation

use switch statement to get user input for what variable to modify and then record that input into the respective variable

use while statement for input validation in each switch case.

Use rand function to get random starting location

}

playAgain function {

exactly the same as start menu function except for minor cout changes

}

}

Ant class file {

This file will hold all the functions needed to change the characters on the board to show movement of the ant and store all the variables properly

Enum group – to store direction of ant for next move
Private variables: direction, previousRow, previousColumn, currentRow, currentColumn, currentColor

Default constructor – does nothing
Destructor – runs silently
Constructor with 2 parameters – receives startRow and startColumn variables and sets Them equal to all set functions accordingly

setDirection function – sets direction = d;
getDirection function – returns direction;

setPreviousRow function – sets sets previousRow = pr;
getPreviousRow function – returns previousRow;

```
        setPreviousColumn– sets previousColumn = pc;
        getPreviousColumn function function – returns previousColumn;

        setCurrentRow function – sets currentRow = cr;
        getCurrentRow function – returns currentRow;

        setCurrentColumn function – sets currentColumn = cc;
        getCurrentColumn function – returns currentColumn;

        setCurrentColor function – sets currentColor = ccc;
                holds the character for white or black space
        get currentColor function – returns currentColor;
        }

main file {
        create variables needed

        call startMenu function
                prompts user to start program or quit

                if user starts program
                call get values function {
                        adjust variables for border
                        create object with arguments
                        create 2d array
                        fill board and create borders
                        create ant
                        print board to screen


                        for based loop that runs for amount of steps
                                call movement function
                                call sleep function
                                call clear screen to show ant movement

                        delete board array

        while loop playAgain function to give user chance to run program infinitely

movement function (passes variables via reference to update them in main function)
 {
        if the currentColor = ' '
                then set previous board location to '#'
                if location = north
                        set direction to east
                        set *startColumn to getCurrentColumn
                        add 1 to *startColumn

                        if *startColumn == 1
```

```
                    set *startColumn = 1 to avoid the border

                setCurrentColumn to *startColumn
                if board's currentRow and currentColumn == ' '
                    setCurrentColor = ' '
                else setCurrentColor = '#'

                board's currentRow and currentColumn = antLocation = '*'

        else if location = east
                repeat all previous lines but adjust accordingly

        else if location = south
                repeat all previous lines but adjust accordingly
        else if location = west
                repeat all previous lines but adjust accordinly



if the currentColor = '#'
        then set previous board location to ' '

        if location = north
                set direction to west
                set *startColumn to getCurrentColumn
                subtract 1 from *startColumn

                if *startColumn == 0
                    *startColumns = columns -2 to adjust for the borders

                setCurrentColumn to *startColumn
                if board's currentRow and currentColumn == ' '
                    setCurrentColor = ' '
                else setCurrentColor = '#'

                board's currentRow and currentColumn = antLocation = '*'

        else if location = east
                repeat all previous lines but adjust accordingly
        else if location = south
                repeat all previous lines but adjust accordingly
        else if location = west
                repeat all previous lines but adjust accordingly

Print out updated board with a double for statement

Set previousRow and previousColumn to current location

}
```

clearScreen function {
clears screen with newline characters to show movement of ant on screen in a smoother fashion
}

Test Case	**Input Values**	**Driver Functions**	**Expected Outcomes**	**Observed Outcomes**
If getCurrentColor=' '	currentRow, currentColumn	setCurrentColor('#'	Board[getPrevRow][getPrevColumn] = ' # '	Board[getPrevRow][getPrevColumn] = '#'
If getCurrentColor='#'	currentRow, currentColumn	setCurrentColor('#')	Board[getPrevRow][getPrevColumn] = ' '	Board[getPrevRow][getPrevColumn] = ' '

Test Case	Input Value	Driver Function	Expected Outcomes	Observed Outcomes
If *startColumn == columns -1	startColumn	Void movement()	*startColumn = 1	*startColumn = 1
If *startRow == rows - 1	startRow	Void movement()	*startRow = 1	*startRow = 1
If * startRow == 0	startRow	Void movement()	*startRow = rows - 2	*startRow = rows - 2
If *startColumn == 0	startColumn	Void movement()	*startColumn = columns - 2	*startColumn = columns - 2

Test Case	Input Value	Driver Function	Expected Outcomes	Observed Outcomes
If currentColor == ' ' && direction == north	direction	setDirection(east)	*startColumn += 1	Moves ant up one column (This is the same thing that was expected to happen)

If currentColor == ‘ ‘ && direction == east	direction	setDirection(south)	*startRow += 1	Moves ant to the right one space (This is the same thing that was expected to happen)
If currentColor == ‘ ‘ && direction == south	direction	setDirection(west)	*startColumn -= 1	Moves the ant left one space (This is the same thing that was expected to happen)
If currentColor == ‘ ‘ && direction == west	direction	setDirection(north)	*startRow -= 1	Moves the ant down one space (This is the same thing that was expected to happen)
If currentColor == ‘#‘ && direction == north	direction	setDirection(west)	*startColumn -=1	Moves the ant left one space (This is the same thing that was expected to happen)
If currentColor == ‘#‘ && direction == east	direction	setDirection(north)	*startRow -= 1	Moves the ant down one space (This is the same thing that was expected to happen)
If currentColor == ‘#‘ && direction == south	direction	setDirection(east)	*startColumn += 1	Moves the ant right one space (This is the same thing that was expected to happen)
If currentColor == ‘#‘ && direction == west	direction	setDirection(south)	*startRow += 1	Moves the ant up one space (This is the same thing that was expected to happen)

Reflection

I learned a lot from writing this program. I have a much better grasp on how to work with objects and why they are useful. I understand how to use pointers even more. While using the object to update the movement of the ant and the board I was able to remove a lot of duplicate code because I realized it was repeating in the

several of the if and else if statements. This was a design change that I was able to make and made my code much shorter and clean.

I ran into several problems. The first major problem I had was how to create the borders. I really just had to think about it for a while and I also had to adjust the user given inputs to create space for the border while still keeping the proper values of the board and location. The second problem I ran into was when the ant would run into the border. It was a simple variable miscalculation I was performing in an if statement that I was able to solve in about 20 minutes. A minor problem I had is I noticed random characters being output to the right side of the board. I had to use the cin.clear() function to fix this. My guess is that it's a problem on the stack that I am not able to see? Lastly, when I was writing the void movement function I literally just had to stare at the screen for close to an hour and put together in my head how I would get the ant to move and update the board. It kind of all just came together as I was writing it.

After writing this program I feel like I understand C++ so much better. I came to the realization that code is about 50% using logic and math to solve the problems at hand and the other 50% is knowing the syntax and library/templates/functions available for you to use in the language.