Christopher Merrill
CS162
10/9/2016

Assignment 1 Reflection Document
(Design, Test Plan, Test Results and Reflection)

DESIGN:

OVERVIEW

I plan on containing a main game loop within an “Ant” class that contains the logic that controls the ant’s behavior. I will then make a Board class that contains the tile information of the game board (whether or not a tile is white or black). The main function will first ask the user all of the necessary parameters (game board size, number of steps the ant will take, and starting location), then instantiate the Ant class, and finally run the main game loop within the ant class.

My plan for the game loop is to use a series of if/then statements that will determine each move the ant makes. The two main if/then statements will be “If the current tile the ant is standing on is white, do…” and “Or else if the current tile the ant is standing on is black”. Within each of those statements will be four more statements that correspond to the direction the ant is facing at that particular step.

Boundary scenario: In the case that the ant moves off the edge, I will use a wrap-around technique. Meaning if the ant moves off the edge, it will reappear on the opposite end of the game board and continue moving. It can be thought of as the ant “tunneling” to the other side of the board.

PSEUDOCODE

MAIN.CPP:
Ask the user to input the number of rows in the array
Ask the user to input the number of columns in the array
Ask the user how many steps the ant will take
Ask the user if they’d like to specify a starting location

If the user wants to specify a starting location:
    Ask the user for a starting row
    Ask the user for a starting column
Else:
    Use a random number generator to choose a random starting row or column

Instantiate the ant class

Run the play function (see below)
End

ANT CLASS
*The Ant Class should only need one function, the play function.
*It will need variables for: The ant's row/column position, the ant's current direction, the ant's current step, the total number of steps, the pause time between frames.

Play Function:
Create the game board
Add the ant to the game board
Print the game board

Main Game Loop:
Do the following while the number of steps is less than the number of steps the user specified:
    If the tile the Ant is currently standing on is blank:
        If the Ant's current direction is North:
            Get the Ant's next location (+1 to columns)
            If the new location is within the bounds of the array:
                Change the value of the current tile (to #)
                Change the ants direction (to East)
                Set the Ant's new location (+1 to columns)
                Update the board with the ant's new position
                Wait ½ second
                Clear the current game board
                Print the updated game board
                Add one to the Ant's steps
            Else:
                Repeat the above code but change the ant's new location to the other side of the board.
        If the Ant's current direction is East:
            Get the Ant's next location (+1 to rows)
            If the new location is within the bounds of the array:
                Change the value of the current tile (to #)
                Change the ants direction (to South)
                Set the Ant's new location (+1 to rows)
                Update the board with the ant's new position
                Wait ½ second
                Clear the current game board
                Print the updated game board
                Add one to the Ant's steps
            Else:
                Repeat the above code but change the ant's new location to the

other side of the board.

If the Ant's current direction is South:

Get the Ant's next location (-1 to columns)

If the new location is within the bounds of the array:

Change the value of the current tile (to #)

Change the ants direction (to West)

Set the Ant's new location (-1 to columns)

Update the board with the ant's new position

Wait ½ second

Clear the current game board

Print the updated game board

Add one to the Ant's steps

Else:

Repeat the above code but change the ant's new location to the other side of the board.

If the Ant's current direction is West:

Get the Ant's next location (-1 to rows)

If the new location is within the bounds of the array:

Change the value of the current tile (to #)

Change the ants direction (to North)

Set the Ant's new location (-1 to rows)

Update the board with the ant's new position

Wait ½ second

Clear the current game board

Print the updated game board

Add one to the Ant's steps

Else:

Repeat the above code but change the ant's new location to the other side of the board.

If the tile the Ant is currently standing on is "#":

If the Ant's current direction is North:

Get the Ant's next location (-1 to columns)

If the new location is within the bounds of the array:

Change the value of the current tile (to blank)

Change the ants direction (to West)

Set the Ant's new location (-1 to columns)

Update the board with the ant's new position

Wait ½ second

Clear the current game board

Print the updated game board

Add one to the Ant's steps

Else:

Repeat the above code but change the ant's new location to the

other side of the board.

If the Ant's current direction is East:

Get the Ant's next location (-1 to rows)

If the new location is within the bounds of the array:

Change the value of the current tile (to blank)
Change the ants direction (to North)
Set the Ant's new location (-1 to columns)
Update the board with the ant's new position
Wait ½ second
Clear the current game board
Print the updated game board
Add one to the Ant's steps

Else:

Repeat the above code but change the ant's new location to the other side of the board.

If the Ant's current direction is South:

Get the Ant's next location (+1 to columns)

If the new location is within the bounds of the array:

Change the value of the current tile (to blank)
Change the ants direction (to East)
Set the Ant's new location (-1 to columns)
Update the board with the ant's new position
Wait ½ second
Clear the current game board
Print the updated game board
Add one to the Ant's steps

Else:

Repeat the above code but change the ant's new location to the other side of the board.

If the Ant's current direction is West:

Get the Ant's next location (+1 to rows)

If the new location is within the bounds of the array:

Change the value of the current tile (to blank)
Change the ants direction (to South)
Set the Ant's new location (-1 to columns)
Update the board with the ant's new position
Wait ½ second
Clear the current game board
Print the updated game board
Add one to the Ant's steps

Else:

Repeat the above code but change the ant's new location to the other side of the board.

Notify the user that the ant has finished its route
End

BOARD CLASS

*The board class will need the following functions:
-Create the game board
-Print the current game board
-Get/Set a particular tile's value
-Get/Set the size parameters of the array

*It will need variables for: The game board and the size parameters of the game board (rows, columns)

Create Game Board Function:
Dynamically allocate a 2D array
Initialize all values to be blank

Print Game Board Function:
Loop through each value of the board and print the value
When the end of a line is reached, start a new line

TEST PLAN and TEST RESULTS:

Test	Results	Comments
Check that the array is the same size that the user specifies. Try: 1x1 3x3 10x10 25x25	Correctly printed out 1x1 3x3 10x10 25x25 Arrays	No changes necessary
Check that the ant moves the correct number of steps. Try: 1, 10, 20, 100 steps and print the result at the end	0, 9, 20, 100	Changed the operator in the while loop to include the last step. The issue was resolved.

Check starting position when the user specifies. Try (0,0), (10,10) (20,20)	(0,0), (10,10), (20,20)	No changes necessary
Check the random starting location is working properly.	The same “random” location is chosen each time	Realized that I forgot to seed the random number generator when using rand(). Added a line for srand() and the issue was solved.
Make sure the ant is moving properly by holding the starting position constant and watching the first 10 moves (change the pause time to 1 move per second)	The ant is repeating the same small loop, treating all squares as if they were black.	My if/then statement that checks if the space is black or white runs the “else” portion (the black tile option) as default if the square isn’t white. I realized each square had the value of the ant when the next move was determined, so it was defaulting to assume that the ant was always a black square. To solve this I implemented two new lines of code in my play() function for each possibility. Before adding the ant to a square I temporarily save the new tile’s position in a new variable. I then add the ant, print the board, and then switch the tile back to its original value before performing the next operation.
Check the boundary case (where the ant moves off the edge), by starting the ant on each edge.	The ant successfully “wrapped-around” for each of the boundary cases. (It appeared on the opposite side of the board)	No changes necessary.
Make sure a pattern emerges after 10,000 steps.	When a large enough array is used, a pattern did emerge around 10,000 steps. (With a small array the ant doesn’t have enough free space for a pattern to emerge)	No changes necessary. (This was expected to work since the ant moved correctly in the 10 step case above)

Input validation for the following user inputs: *Number of Rows *Number of Columns *Number of Steps *Decision to choose a starting location or not. *Starting Row *Starting Column	When inputting a character instead of an integer (or vice versa), the program successfully prompts the user to choose again. It also correctly tells the user to enter an integer above zero when necessary.	Note: I did not include input validation in my original plan. This was something that slipped my mind since I was concerned with the game logic. However I know now to always include input validation when necessary.

REFLECTION:

Although there were a few problems that I wasn't able to prepare for in my original design document, the final result essentially worked as planned. As mentioned in my test plan, I originally had an issue where the ant would repeat the same small loop over and over. It took a long time to figure out that the ant was defaulting to the behavior of a black square on every move. This was due to the fact that my if/then statement that depended on the tile value defaulted to black square behavior if the tile wasn't white. I then realized that at each move, a white tile would actually have the value of "*" (the ant). I had to update my code to revert a tile back to its original value after printing it to the console, so that the ant's behavior would be correct. This was the only real deviation from my original plan, however I also forgot to include a menu function or input validation functions since we rarely had to use them in CS161. I was able to build them and get them to work, but next time I will be sure to add them into my original plan. Overall I felt that my plan was a success since, after all, it worked correctly. However I still feel that my play loop is ridiculously long and I'm curious to know if there is a more efficient way to obtain the correct behavior without a series of if/then statements.