Paul Min
07/07/18
CS-162 Summer 2018

Description: Project 1: This is my design and reflection document. In it I outline my approach to coding Langton's ant.

Design

Programming style and documentation:

I will follow the coding style guidelines provided on Canvas. I will use comments to help document my code. I will check for memory leaks. I will make sure to de-allocate any dynamic memory that was allocated. I will validate input when necessary. I will make sure to submit all my files in a zip archive.

Build the board using dynamic 2D array:

I will use the same strategy employed in lab 1 to build a dynamic 2d array. The stackoverflow link provided in canvas will serve as a great reference. Instead of a 2x2 or 3x3 matrix, the dimensions of the board will be determined by user input.

To implement this, I plan on having a separate board class. The board class will have a constructor that will accept two arguments (an integer value for the row and an integer value for the column).

Input validation functions:

I will create input validation functions that prevent the user from entering zeros/negative numbers to be passed in as the rows and columns. I will have functions that account for characters/letters. I will make sure that my input validation prevents an endless loop. To do this I plan on using std::cin.clear() and std::cin.ignore() functions. I will reference stack overflow to implement best practices.

For my input validation functions, I plan on using bool flags and do while loops to structure my input validation. For example here is some pseudocode that shows what I'm thinking:

```
Set bool flag to false
do
  Get input from user
  If input does not return errors and input is either int 1 or 2
        Set bool flag to true
  else
```

clear buffer
Ignore 256 characters or up to first newline
Prompt user that input was invalid. Please choose between int 1 or int 2
while bool flag is false

For floating point numbers, I've decided to allow them as they will get truncated to an integer. So if a user enters 12.5 for the number of rows, the number of rows will be 12.

Create the source and header files for the Ant class properly:

I have a few different ideas on how I will keep track of the ant location. After each move depending on whether it goes up, right, down, or left I will increment or decrement the row or column. To keep track of the ant's orientation I plan on either using an Enum or constant integer variables with up = 0, right = 1, down = 2, left = 3. I plan on having functions called turnLeft() and turnLeft() that will change the orientation of the ant, and a function called moveForward() that will move the ant forward based on its orientation.

For the Ant class function specification file, I plan on having several private data members. I think I will have private data members for the antRow, the antCol, the antDirection, and lastly I will have a pointer to a board object.

I plan on having the following member functions: setup, setAnt, play, and a destructor. The setup method will have two integer parameters for boardRow and boardCol respectively. This method will dynamically create a new board object. When a board object is created, it should dynamically create a 2d array based off the boardRow and boardCol that was passed to the setup Ant method. It will also initialize all elements to a blank space to represent a white board.

The setAnt method will place the ant on the board. Immediately after placing the ant, the method should print the board. And immediately after printing the board, the method should set the piece back from * to empty space(white).

The play method will contain a for loop that will loop until the number of steps specified by the user is reached. Within the loop, there will be two main conditionals. The address to possible cases in this simulation.

Case 1: If piece is white

Case 2: If piece is black

The destructor will delete the pointer to the board object.

Print each step correctly, including running for the correct number of steps:

I will create a print function that prints out the board and shows which spaces are white, which spaces are black, and which space holds the ant.

To keep track of the steps, I will use a for loop that will print out the current step count during each iteration. According to wikipedia, Langton's ant starts developing a recurring highway pattern after 10,000 steps. I will place my maximum step limit at 100,000 so that the "highway" is visible. I also plan on making the minimum number of steps 1.

The printing of each step allows the user to see the changes in the shape:

Similar to the previous criteria, I will use a for loop that will iterate for the number of steps specified by the user.

For each iteration of the loop and until the number of steps specified is reach, I will check whether the ant is on a white space or a black space. The ant will follow 2 rules:

- If the the ant is on a white space, it's orientation will change 90 degrees to the right and the space will change to black. Afterwards, it will move forward one space based off its orientation. After the ant is moved forward I will print the board. Immediately after the board is printed, I will set the piece back to the color of the piece that the ant is on.
- If the ant is on a black space, it's orientation will change 90 degrees to the left and the space will change to white. Afterwards, it will move forward one space based off its orientation. After the ant is moved forward I will print the board. Immediately after the board is printed, I will set the piece back to the color of the piece that the ant is on.

Allow the user to specify the starting location of the ant:

I will attempt to let the user choose a random starting point, which will place the ant within the bounds of the dynamically created array. I will limit the range of random numbers to fall within the bounds of 2d dynamic array that represents the board. To implement random placement, I plan on using a random generator seed, the srand function, and the time function. I will reference chapter 3 section 10 in the textbook for guidance.

Implement a menu function that can be reused in later program:

I plan on creating an interface class that will serve as the glue that brings together the ant and board classes to make this program work. The interface class will not have any private data members. Instead it will have public member functions. The functions will handle prompting the user for input, validating their input, and deciding what to do based off their input. I will have functions that display the starting menu, validating the starting menu choice, prompt the user for the number of board rows, validate the user input for board rows, prompt the user for the number of board cols, validate the user input for board cols, prompt the user for the number of steps the simulation should run for, validate the user input for the number of steps, prompt the

user where to place the ant, validate the user input for ant placement, a function that packages prompt and validate functions together, a function that prompts for random placement, a function that validates random placement choice, two functions to randomly generate row and column, and a display ending menu function.

Test table

Test Case	Input Values	Driver Function	Expected Outcomes	Observed Outcomes
Input too low	Input < 1	validateStartingMenuChoice()	Invalid input. Please choose between the integer ‘1’ or the integer ‘2’	Invalid input. Please choose between the integer ‘1’ or the integer ‘2’
Input too high	Input > 2	validateStartingMenuChoice()	Invalid input. Please choose between the integer ‘1’ or the integer ‘2’	Invalid input. Please choose between the integer ‘1’ or the integer ‘2’
Input in correct range	2	validateStartingMenuChoice()	The End.	The End.
Input in correct range	1	validateStartingMenuChoice()	Would you like to randomly place the ant? Enter the char ‘Y’ for yes or ‘N’ for no:	Would you like to randomly place the ant? Enter the char ‘Y’ for yes or ‘N’ for no:
Input not a character	1, 10.5, “y”	validateRandomPlacement()	Invalid input. Please enter a char ‘Y’ or a char ‘N’	Invalid input. Please enter a char ‘Y’ or a char ‘N’
Correct character input	‘Y’ or ‘y’	validateRandomPlacement()	Should call promptNumRows()	Calls promptNumRows()
Correct character input	‘N’ or ‘n’	validateRandomPlacement()	Should call promptNumRows()	Calls promptNumRows()

Input too low	Input < 2	validateNumRows() validateNumCols()	Invalid input. Please enter an integer between ‘2’ and ‘400’	Invalid input. Please enter an integer between ‘2’ and ‘400’
Input too high	Input > 400	validateNumRows() validateNumCols()	Invalid input. Please enter an integer between ‘2’ and ‘400’	Invalid input. Please enter an integer between ‘2’ and ‘400’
Input in correct range	400	validateNumRows() validateNumCols()	The number of rows is: 400 The number of cols is: 400	The number of rows is: 400 The number of cols is: 400
Input too low	Input < 1	validateSteps()	Invalid input. Please enter an integer between ‘1’ and ‘100,000’	Invalid input. Please enter an integer between ‘1’ and ‘100,000’
Input too high	Input > 100,000	validateSteps()	Invalid input. Please enter an integer between ‘1’ and ‘100,000’	Invalid input. Please enter an integer between ‘1’ and ‘100,000’
Input in correct range	100000	validateSteps()	Starts the simulation printing through 100,000 steps	Starts the simulation printing through 100,000 steps
Input not a character	0, 1, 10.5,	validateEndingMenu()	Invalid input. Please enter a char ‘a’ or a char ‘b’	Invalid input. Please enter a char ‘a’ or a char ‘b’
Input not correct character	Input != ‘a’ or ‘A’ or ‘B’ or ‘b’	validateEndingMenu()	Invalid input. Please enter a char ‘a’ or a char ‘b’	Invalid input. Please enter a char ‘a’ or a char ‘b’
Correct character input	‘A’ or ‘a’	validateEndingMenu()	Prompt random placement	Prompt random placement
Correct character input	‘B’ or ‘b’	validateEndingMenu()	goodbye!	goodbye!

Reflection

Looking back, this has probably been the most complicated C++ project I have coded so far in this program. I remember reading the project specifications and feeling so overwhelmed at how complex this project seemed. I think the biggest breakthroughs for this project came after visualizing that there are only ever 2 cases in this program. The first case being that the ant is on a white space and the second case being that the ant is on a black space.

During the initial stages of my research, I saw someone post a piazza post about watching numberphile's langton's ant video on youtube. I watched that video and similar videos to it, and it gave me the idea to create separate functions to change the orientation of the ant and to move the ant forward. Breaking down the project into smaller pieces really helped me tackle this assignment.

In the beginning of my project I tried to create one big function that would change the ant's orientation and move the ant forward, but I realized that it would be better to break it apart into separate smaller functions.
One problem that I encountered was figuring out how I would use just one dynamic 2d array of chars to represent both the color of the board and the position of the ant. I somehow had a breakthrough moment, when I realized I can set the ant on the board, print the board, and the set the board back to the color that ant is on. Doing this allowed me to just have one 2d array, thus making my program more efficient than having 2 2d arrays.

Lastly, I learned that diving head first into code without a design is a bad idea. Figuring the logic and designing the projects is where the complexity lies. Coding the syntax is relatively straightforward.