CS 162 - Project 1: Design + Reflection Document
Jeovani Vela
1/20/18

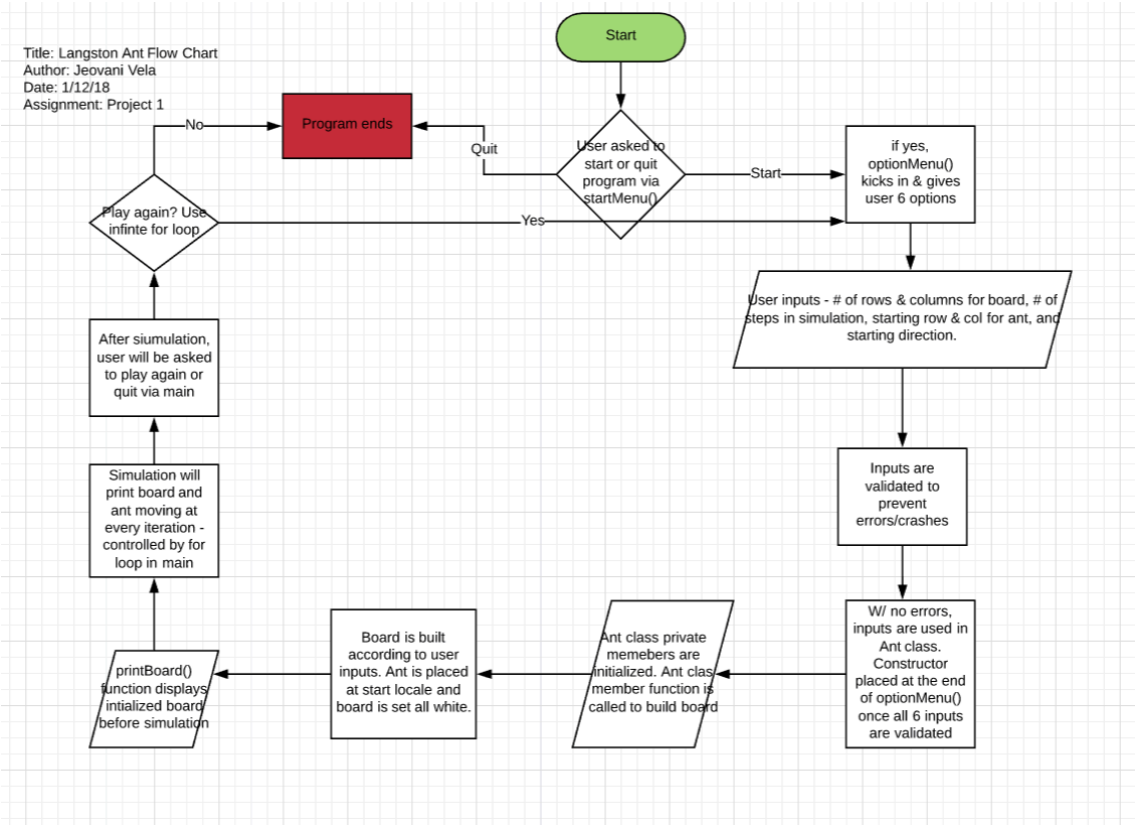
This project was based on the implementation of Langston's Ant. The rules set forth also happened to serve as some of the problems to be solved when building the program. It was very tedious to build and took much time diagramming out.

Requirements:

-Langston's Ant is placed on a 2x2 board that is filled white and starts moving forward, with the exception of two rules:
  1) If ant is on a white space, turn right 90 degrees and change the space to black.
  2) If ant is on a black space, turn left 90 degrees and change the space to white.
-Display a menu to the user to choose options in order to fill data needed for simulation
-The program will take in user inputs to help fulfill the ant simulation.
  -Must get the number of rows and columns, no less than 3x3 to build board.
  -Must get the starting row and column. Must not exceed parameters of initial entry.
  -Must get number of steps to be performed and starting direction to determine next step.
-Must validate inputs by user.
  -Inputs from user must be validated to avoid potential program crashes or faults
-Must use a class called Ant that should contain all the information of the board, including the ant's location and the ant's orientation.

Design:

First present user with the option to start or quit the program. If user decides to quit, program ends. If user decides to start, the user will be presented with a menu giving him/her the option of 6 choices to input data required for the simulation. All inputs are validated. Once validated and all 6 inputs are filled, the menu will contain a function to call a constructor from the class Ant to make an Ant object. Once in the Ant constructor, an Ant object is created based on the user inputs. Plus there is a private member function within the constructor to make the board. It



is easier to code in this format as it remains in scope. The board is built according to user specifications, including number of rows and columns, starting row and column and the direction. Plus the board is initialized to all white spaces. The board is then printed followed by a series of print outs based on the number of steps input by the user. In the main file, a class member function gets the number of steps and uses them in a loop to control the iterations and present the board with the ant movements. Lastly it will ask the user if they would like to play again or quit via an infinite loop.

Test Plan:

Test Case	Input Values	Driver Functions	Expected Outcomes	Observed Outcomes
Start screen - user inputs series of letters and numbers instead of 's' or 'q'.	`32141234, hello, –100, aaaa, 7hdfg, 5435fgsfgz,`	startMenu Function - accepts only a 's' or 'q'. Lower/uppercase does not matter. Input is validated.	Program will not begin until either 's' or 'q' is entered in the screen by the user. Will ignore numbers of any sort/ sequence.	Program will not begin as the series of inputs are invalid. Program will only begin if 's' or 'q' are entered by the user.
optionMenu function- user is asked to enter the number of rows for board	`–100, 0, 1, 2, dsafdsf, 3`	optionMenu Function - Program has a set min on the number of rows for the board	Program will not validate any inputs less than 3. Including any letter combinations.	Program does not validate any inputs until the number 3 is entered. (Any number greater than 3 for that matter)
optionMenu - user is asked to enter the number of columns for board	`–9, 0, 2, sfdh, 5`	optionMenu - Program has a set min on the number of cols. Cannot be less than 3 or than the number of rows	Program will not accept anything less than 3 or less than the number of rows on the board.	Program does not crash. Instead it disregards the input until a valid input of 5 is entered.

optionMenu - user is asked to enter the starting row or col	`(Board=5x5)`
	
	`88, –100, –1, 0`
		optionMenu Function - Program has s set min on the starting row/col. Cannot exceed boundary of board	Program will discard any value less than the number of rows or columns	Program disregards all values until the starting row is entered a value of 0. This includes testing the starting column, option #5.

optionMenu - user is asked which direction to start ant: up, down, left or right	`4352435, sdfgsd, hello, tease, up`	optionMenu Function- Program validates each input to match directions.	Program fails to continue until one of the four (up, down, left, right) are entered.	Program does not start until the user enters up, which matches with the validation input for that particular option.
In main, Ant class member function is called to get the num of steps input by user	`Input is validated 1st and received from Ant class`	Ant class members function gets the num of steps for the simulation and brings them to the main to be set to a variable that will be used to control iterations	getNsteps(); function properly gets and returns the correct value. And is able to be initialized to an int variable to be used in a for loop.	getNsteps(); function properly gets and returns the correct value. And is properly used with an int variable for a for loop.
Main - user is asked after simulation to play agin	`and, 11111, odaf989, h, q`	In the main, an infinite for loop ask user to play again - choose either 's' or 'q'	Program will not cooperate with user until one of the two accepted char values are entered.	Program does not cooperate with the user until a valid char is entered. 'q' is entered and the program quits as expected.
optionMenu - user is asked to input starting row and starting col	`1.0`	In the optionMenu, the user inputs a floating number for both the starting col and row, 1.0 on each	Program will not accept the 1.0 and move on, and later result in a crash or fault.	Program was still able to properly place the ant at row 1, col 1 on the board.
optionMenu - user inputs a double for all input	`User inputs a series of doubles, 5.0,5,0 for board. Starting row and col at 1.0`	In the optionMenu, the user inputs a floating number for some of the options giving in the menu	Program will crash and not accepts as many double values.	Program was still able to execute properly as board built ok and ant placed at correct location. Also the correct number of steps were performed.

Reflection:

For this project building the menu was not as complicated. Based on my previous c++ knowledge of loops and if statements, it worked out fine. I decided to use that format over switch statements as I was able to use a counter to keep track of my iterations. I did struggle

finding a way to initialize my member variables in the Ant class until I decided to play a constructor in one of menu functions. It helped me transition through my code more sufficiently and I was able to keep track of my steps more accurately. Most importantly, having an Ant class object made it easier to keep track of the ant and its placement. I initially thought of making my ant a char *, but soon realized after I made my board I could include the placement of my ant thereof. Additionally, I utilized a get function to help keep track of my iterations in the main file so that my board and ant are moved accordingly. I did struggle trying to find a way to give the user the option to play again or quit. I used while loops and do while loops various ways until I was able to use an infinite for loop. It helped not interfere with the first part of my program and allowed the user the option to continue or quit. Additionally, although I created a flow chart to help me compartmentalize writing my code I did struggle with some of the logic of the board and the ant movement. Finally I realized the subscripts would be my best friend in this method and used them properly. Overall this project was a struggle for me, however after reworking my code in certain areas and debugging it overtime, I can admit it has helped me realize and clear up some of my shortfalls.