Peter Moldenhauer
moldenhp@oregonstate.edu
9/29/16
CS 162

Assignment 1 – Reflections Document

Design description:

Immediately after reading the assignment description, the very first step I took was to get out a pencil and paper and write out a brief description of the problem that was to be solved for this assignment:

You will design, implement and test a program that implements a simulation of Langton's Ant. This can be considered a cellular automation (have an array or matrix of cells). Each turn or step the value of each cell may change based upon a simple rule. For the Ant there are two simple rules. 1) In a white square, turn right 90 degrees and change the square to black. 2) In a black square, turn left 90 degrees and change the square to white.

After doing a little research on what exactly is Langton's Ant and looking at some of the specific requirements based in the assignment description, I was able to make a thorough list of things I needed to do so solve the problem. This list aided in my understanding of the problem and shed some light on the specifics of what I needed to accomplish for this assignment:

- I was to build a program that simulated Langton's Ant
- I need to get data that is inputted from the user
- I needed to use data entered in by user to create a 2D array "board"
- I needed to create an "ant" that appeared on the board
- I needed to use data entered in by user to place the ant in a random location or a specified location from user
- I needed the ant to "move" around on the board in a very specific manner.
- I needed the ant to change the color of the board based on how the ant moves on the board
- I needed the ant to keep moving on the board for the number of steps based on user input
- I needed to output to the screen every step that the ant takes on the board.

I then took what I wrote about my own interpretation of the problem and then compared it with the specific requirements that were to completed as stated in the assignment description. I then broke up the assignment criteria into numerous smaller assignments. (Similarly, when it came time to code the program, I broke up the program into smaller code blocks that I would write and test write away.) By breaking up a large assignment into multiple smaller assignments it is easier to focus on only one or two problems at a

time that I need to complete instead of trying to tackle everything all at once. The smaller "assignments" that I broke up the large assignment into are as follows:

- Figure out Langton's Ant algorithm
    - Create pseudo code for the algorithm
    - If current color is white what happens?
    - If current color is black what happens?
- Create pseudocode for entire program to demonstrate how the program should look
    - From program start to program finish what will take place
- Plan and code user input in program
    - Create variables
    - Create functions to validate data entered by user
    - Create random number function
    - Test code
- Plan and code 2D array in program and print it to the screen
    - 2D array had to be dynamically created
    - Need to delete array at end of program to free up memory
    - Test code
- Plan and code an Ant class and place the Any symbol on the 2D array board.
    - For testing, use same location on the array each time, not a random spot
    - Ant class need to contain various info (direction, current color etc.)
    - Test code
- Plan and code Langton's Ant algorithm into program and create a loop to implement it
    - Use short loops/system pause command to test algorithm to see if its working properly
    - Test code
- Plan and code the program so it does not crash when the ant "steps out of bounds"
    - Dynamically create a new array/copy old array/delete old array?
    - Instead of creating new array, just jump to opposite side of board to create an "endless" board?
    - Test code
- Finishing touches on program
    - Delete any unnecessary comments
    - Final formatting of output

In my step of "figuring out Langton's Ant algorithm" I took the two simple rules that were given in the assignment description and converted them to pseudo code. This pseudo code mostly contained if statements. I assumed that the ant would be starting on a blank space and that the ants starting direction would be North. The pseudo code that I created is as follows:

If the ant is on a white space then:

- Change space to black once ant moves (blank space becomes #)
- Change direction ant is facing

    - North becomes east
    - East becomes south
    - South becomes west
    - West becomes north
- Change the 2D array to hold the ant symbol in the new space
    - If new direction is east then the ant's row number should add 1
    - If new direction is south then the ant's column number should subtract 1
    - If new direction is west then the ant's row number should subtract 1
    - If new direction is north then the ant's column number should add 1

If the ant is on a black space then:

- Change space to white once ant moves (# becomes a blank space)
- Change direction ant is facing
    - North becomes west
    - East becomes north
    - South becomes east
    - West becomes south
- Change the 2D array to hold the ant symbol in the new space
    - If new direction is east then the ant's row number should subtract 1
    - If new direction is south then the ant's column number should add 1
    - If new direction is west then the ant's row number should add 1
    - If new direction is north then the ant's column number should subtract 1

In my step of "plan and code user input in program I made a note as to what prompts I should give to the user which will then result in getting what data. The prompts given to the user and data taken from the user is as follows:

- Prompt user for number of rows to make the board [get numRows value as input]
- Prompt user for number of columns to make the board [get numCols value as input]
- Prompt user for number of steps the ant should take [get numSteps as input]
- Prompt user to determine if a random starting location for the ant is desired
- If the user does not want a random starting location, prompt user for the starting row location [input row] and starting column location [input col].

Overall, through the process of implementing my design plan, the ultimate goal is to display output to the user a simulation of Langton's Ant. With that being said, with all of the input prompts coded correctly, the 2D array designed correctly and the algorithm thought out properly, I still had to make sure the user could actually see the ant taking steps and moving around on the board. To accomplish this I needed to print out the board and then clear the screen and print out a new board in the exact same place of the old board. This new board will have a new location of the ant stored in it and will make it appear to the user that the ant is moving. This visual display of the ant moving around on the board and leaving various white/black spaces behind it will be the true test to see if I created a successful simulation of Langton's Ant.

Test plan:

My overall testing strategy for this assignment was to use “incremental development”. More simply, I proceeded to test my program continually as I wrote the code for it. I got one part of my program written out and then I tested it right away. If the program compiled and ran properly, I would then proceed to write another part of the program. If this part of the program did not compile and/or run properly, I would work out the bugs in this one small segment of code before even starting on writing other parts of the program.

My plan was that by using this incremental development strategy, a few things would work in my favor. First, by writing only small portions of code at a time and testing right away, if an error did exist then I would only have to look at small blocks of code at a time to fix the errors. If I wrote out the entire program before testing, I am sure it would be very time consuming and frustrating going through the entire program top to bottom over and over again until I found all of the errors. Second, by continually testing my program over and over again, I not only am able to make sure everything runs properly but also that the output to the user displays as intended. I feel that adding spaces and new lines throughout this testing phase to make the output look presentable was less time consuming versus waiting until the very end to go through the program all over again to format the output properly. Lastly, using incremental development ensured that I had at least some sort of a working program to submit at all times. If something were to happen in which I did not have enough time to finish the program before the due date, at the very least I could submit what I have already completed. In this situation, my program may not be complete but I know that it runs properly so I would most likely be able to receive at least partial credit for the assignment.

As I was testing my program I also tested it with the mindset of someone that was intentionally trying to get the program to crash or have erroneous results. By doing this I can make sure that I validated any and all possible options of data entry. This makes it impossible for the program to run on bad data by only accepting data that will cause the program to run properly. Entering in negative numbers, letters and characters for the number of rows, columns and steps was the main way I tested using this mindset.

Lastly, I needed a specific plan to test and see if the ant is actually taking the correct number of steps, moving in the correct direction and leaving being the correct colored space behind. If working properly, the program should keep outputting a new board over the previous board in the same location which makes the ant appear to be moving on the screen. However, this in theory will happen very fast so it will be difficult to watch the path in detail to see if the ant is truly moving correctly. My plan for this issue was to run the program with only two or three steps at a time that the ant will make. This way I would only have to retrace just a small number of steps that the ant made so that I can make sure that the ant moved properly.

Test results:

As much as I was hoping that my program would run perfectly on the first time, of course this was not the case. Due to the fact that I was testing my program throughout the process of coding it, the majority of my program compiled and ran properly right away. However, it wasn't until I started testing the algorithm to get the ant moving around on the board properly when I started to get multiple errors. Of course I had the typical syntax errors here and there such as missing semi colons and misspelled variables that were fairly easy to identify and correct. When my program did compile and run, I could not even see the ant moving around on the board let alone see the ant itself.

I was soon able to get my program working where I could see the ant on the board yet the ant would just keep making a circle on the board. The ant was not actually following the proper steps of Langton's Ant. After some time and frustration of figuring out what the problem was I eventually got the ant to move around on the board properly following the rules of Langton's Ant.

I soon encountered yet another problem during my testing. If I chose to have the board size be rather small and/or have a very large number of steps for the ant to take, the ant would move out of bounds off of the 2d array and then the program would crash. I knew the reasoning why my program was crashing like so but it took even more time to locate and correct the specific cause of this problem.
Finally, after numerous misspelled words, various syntax errors, an incorrect algorithm to move the ant and not properly implementing a method to prevent the program from crashing, I had a program that compiled and ran properly. Even though I still had to work out a lot of problems after going through a rather thorough design phase, I am sure this testing phase would have been a lot more painful if I didn't put so much thought and time into the design.

Comments on how problems were resolved:

I used a variety of methods to go about solving the problems that I encountered during the testing phases of the program. As noted earlier, it was a huge help to write the program in little pieces and test each piece separately and as I wrote them. I only had a few lines of code to go through each time which helped me solve a lot of problems right away. It is a lot easier to fix an error in a 10 line program than an error in a 100 line program. This is a tactic that I will definitely use for testing future programs.

My first big problem that I had with my program was that I could not see the ant moving around on the board, let alone see the actual board on the screen when the program ran. I initially built the board so that it would be initialized with all blank spaces (' ') and I realized that this makes it difficult to even see if the board is being outputted. So in turn, I changed the board being initialized with blank spaces to being initialized with underscore characters ('_') so that there was actually something being outputted to the screen. This was a big breakthrough to me because I then realized that the reason why I was not actually seeing the ant on the board was because I was making the board too big. My

computer monitor was too small and I was creating too man rows/columns for the board. Now with the underscores however, I was able to tell that the board was indeed getting printed out and the ant was indeed on the board, I just had to make the rows/columns smaller.

Now that I could actually see the and moving on the board, I now faced the problem with the ant just moving around in a circle each time the program ran. This obviously meant that the ant wasn't following the specific rules of Langton's Ant. To solve this problem, I initially just used a very small amount of steps that the ant would take on the board and then mentally calculate at what point the ant was making the wrong turn. This was the plan that I came up with in the design phase but it was still challenging to determine the specific cause of the problem. I then came up with another solution that I did not think of in the design phase. My new solution was to use the system("PAUSE") command after each iteration of the loop so that I could specifically control the loop and advance to each step when I wanted to (instead of the program just looping through the steps on its own). The system("PAUSE") command is used on Windows machines and allowed me to control the execution of the program by pushing ENTER to continue the execution of the program. Using this command was a big help because I was specifically able to see how the colors were changing on the board and how the ant's direction changed with each specific step. I was quickly able to conclude that my error was in my if statements on controlling the direction of the ant when it landed on a particular color.
The last big error that I came across with my program was due to the ant moving off of the boundaries of the board which would then crash my program. This would crash the program because the program was trying to execute the command to initialize a space in the 2D array which didn't exist. Even though I thought about this earlier in my design phase, I had completely forgot to take this into consideration when I was actually coding the program. Now realizing that this was a major flaw in the program, I figured out that I could do one of two things to solve this program. The first way to solve this problem was if the ant took a step out of bounds of the 2D array, I would then have to dynamically create a new 2D array (probably twice as big) and then copy the first array to the new array. I would also have to be sure that I was to delete both arrays at the end to free up memory. My second way to solve the problem would be to simply wrap the ant from one side of the board to the opposing side of the board if the ant reached a boundary of the 2D array. This could be coded with simple if statements switching the largest row/column index with the smaller row/column index. Seeing that this second option would be much less time consuming and much less error prone, this is the option that I decided to go with. I always like to follow the KISS mentality (Keep It Simple Stupid) and this second way to prevent the program from crashing was a lot simpler while still following the requirements of the assignment.

The following chart gives a quick description of some of the specific tests to be run throughout the program, the reasoning for the tests and the expected result from the test.

Test number	Test	Purpose	Expected Outcome
1	Input negative numbers at row, column and step prompts	Make sure that the program will not accept an input below 1	Loop to keep prompting user to re-enter a valid input
2	Input letters at row, column and step prompts	Make sure the program will not accept an input that is a letter	Loop to keep prompting user to re-enter a valid input
4	Input special characters and punctuation at row, column and step prompts	Make sure the program only accepts a positive integer	Loop to keep prompting user to re-enter a valid input
5	Input a value other than y, Y, n or N at prompt to have random starting location.	Make sure the program only proceeds with accepting a y, Y, n or N at this prompt	Loop to keep prompting user to only enter one of the valid letters
6	Manually draw out on paper the steps of ant and colors of the board as the ant moves for a certain amount of steps	Compare the hand drawn result with the program result and make sure the two match up for a given amount of steps	The hand drawn board and the program board should match up.
7	Repeatedly run the program with a random starting location for the Ant	Make sure that the ant is indeed starting at a random location with every run of the program	The ant should start in a different location on the board each time the program is run
8	Run the program with the ant starting at an edge of the board	Make sure that the program does not crash when ant moves off the board	If the ant is to take a step that would be off the board, the ant will wrap around to the opposing side and continue on with steps
9	Run program with the ant moving at least two times	Make sure that a new board with the appropriate colors is indeed getting outputted to screen with every step of the ant	The screen should clear off the old board and print out the new board to the screen with every step of the ant.