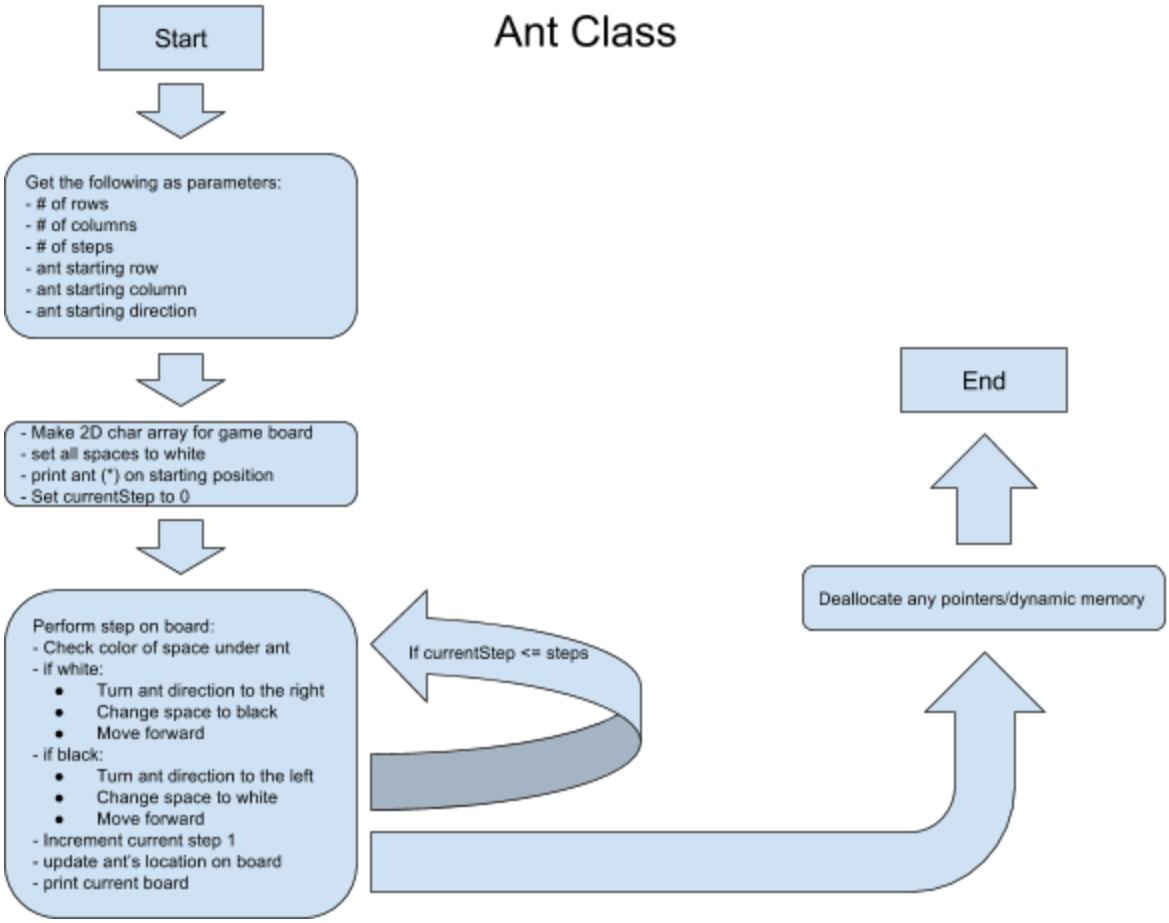
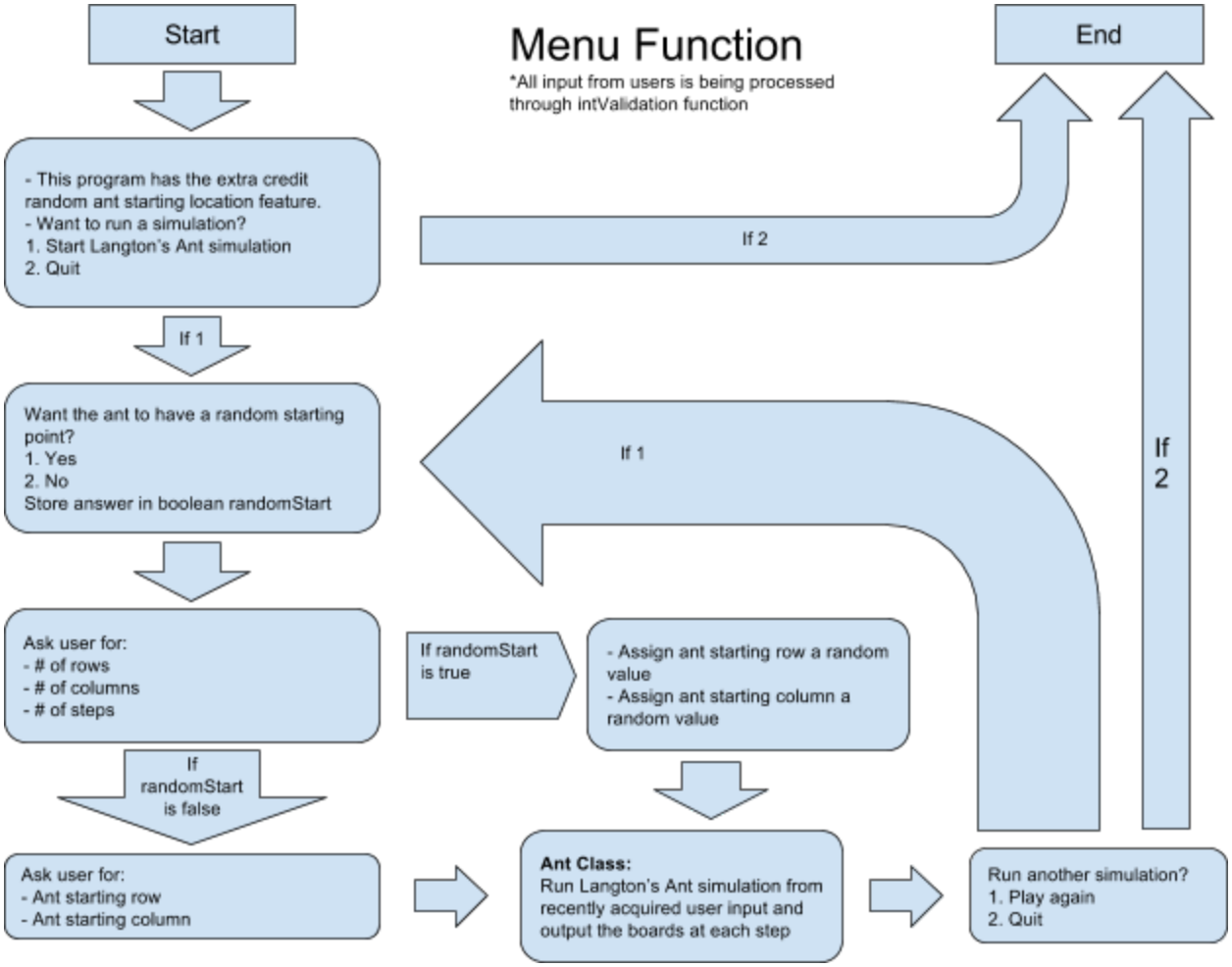
Joel Huffman
Project 1 design document

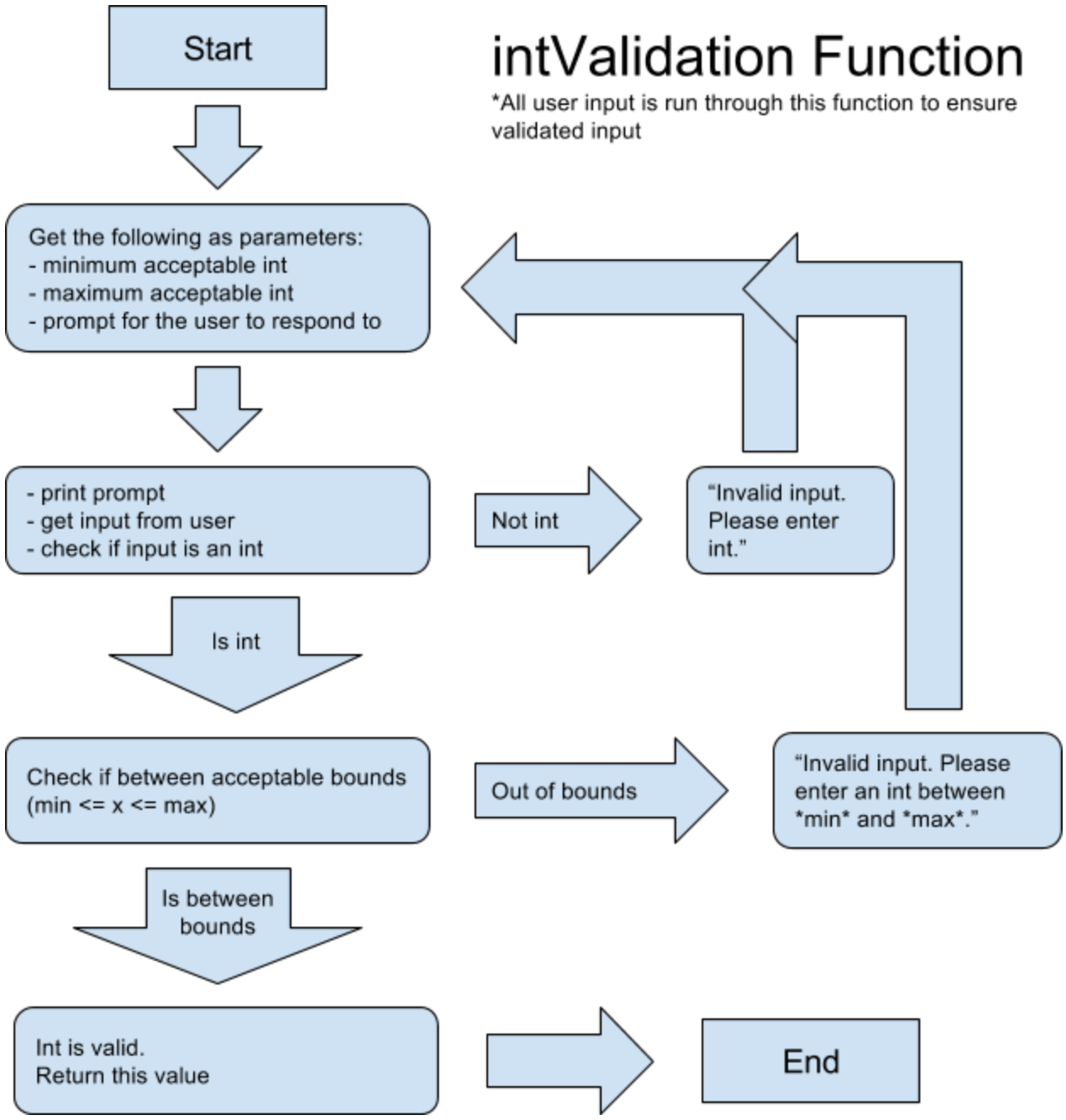
Design:



Joel Huffman
Project 1 design document



Joel Huffman
Project 1 design document



Joel Huffman
Project 1 design document

Test table:

Test Case	Input Values	Expected Outcome	Observed Outcome
Input Validation			
Input too low	-1	Prints “Invalid int. Must be between *min* and *max*” And re-prompts user for input	Prints “Invalid int. Must be between *min* and *max*” And re-prompts user for input
Input too high	99999	Prints “Invalid int. Must be between *min* and *max*” And re-prompts user for input	Prints “Invalid int. Must be between *min* and *max*” And re-prompts user for input
Input wrong type (string)	nine	Prints “Invalid input. Please enter an integer” And re-prompts user for input	Prints “Invalid input. Please enter an integer” And re-prompts user for input
Input wrong type (char)	a	Prints “Invalid input. Please enter an integer” And re-prompts user for input	Prints “Invalid input. Please enter an integer” And re-prompts user for input
Input wrong type (float)	2.3	Prints “Invalid input. Please enter an integer” And re-prompts user for input	Prints “Invalid input. Please enter an integer” And re-prompts user for input
Ant class			
Ant move edge case	----- |* #| |###| |## | ----- Input ot move ant upward next step	----- |# #| |###| |*# | ----- Ant wraps around top and is moves to bottom of board	----- |# #| |###| |*# | ----- Ant wraps around top and is moves to bottom of board
spaceUnderAnt	-----	-----	-----

getting correct data	|* #| |###| |## | ----- spaceUnderAnt = white	|# #| |###| |*# | ----- spaceUnderAnt = black	|# #| |###| |*# | ----- spaceUnderAnt = black

Reflection:

For my design document, I planned out my program through hand drawn flowcharts with some pseudocode written in a few of the steps. I chose to handwrite the flowchart for the beginning of the design process so I could easily erase and alter the steps and methods as I saw design issues and ways to optimize. Once I reached a design I was satisfied with, I converted it into the digital flowchart seen above.

My program largely followed the above designs, but there were some changes I made once I got into the thick of programming. These are outlined below for each individual sourcefile.

The Ant class I wrote had several deviations from my design. Most of these were areas where I had to go deeper into detail than my initial design originally accounted for. A good example is that I didn't account for how I would store what color space the ant is currently on. I thought that I could simply get the character from the ant's current x and y position from the 2D char array, but I soon realized this value is always going to be * because I was using the asterisk symbol to mark where the ant was located every step. I alleviated this issue by using a dedicated char variable (spaceUnderAnt). I would update the contents of this variable to the location the ant was about to move to, before I would draw the * onto the location. This allowed me to have accurate information of about the space underneath the ant.

My design of the Ant class could have benefitted from more time spent mapping out what functions it needed and how to implement them. I ended up breaking the entirety of the take "run a Langton's Ant simulation" into several smaller functions within the class. These functions included: takeStep(), moveForward(), printBoard(), and simulator().

The takeStep() function followed the "Perform step on board" section of my Ant class flowchart, but I separated the section of moving the ant forward into a new function, moveForward(), to give the function more focus and removed it from the if statements because it was called for both forks of the if statement.

The printBoard() function grew from the necessity of having to print the board ever step and calling a function was helpful in the debug process.

The simulator() function was implemented to just continually run my takeStep() functions for every step of the simulation. This could have been included in takeStep(), but I chose to have it be its own function to further compartmentalize the program.

The menu function followed my design pretty closely. I chose not to include separate logic blocks for the validation of each user input simply because it would take up all the room in

the flowchart. I did try to keep the function more objective than I initially anticipated to allow for the reuse (with slight modifications) of the function in future projects.

My intValidation function started pretty faithful to my initial design. I took the input validation code that I made for lab1 and improved it with some help from Long Le's Piazza post about utility functions. My input validation from lab1 was very specialized for that specific situation, so I expanded it into a function and used parameter variables to allow it to be generalized to valid int input for any continuous set. I hope to utilize this function in future assignments for int validation.

Overall I learned quite a bit from working on this project. First and foremost, I started very early on this project and am glad I did. It allowed me to anticipate what sections of the assignment would give me the most issues and attribute my time and effort to them accordingly.

Secondly, I learned that no matter how much you plan ahead, most likely designs and implementations will have to change from unforeseen issues. It's very useful to learn to adapt accordingly to these changes because software development is an agile environment, with changing requirements and needs.

Lastly, this project provided me with a better understanding of how much design creativity coding allows us. While every student of this class has the same requirements for this assignment, I'm sure each submission is unique in a multitude of ways. In much more barbaric terms, there are many ways to skin a cat.