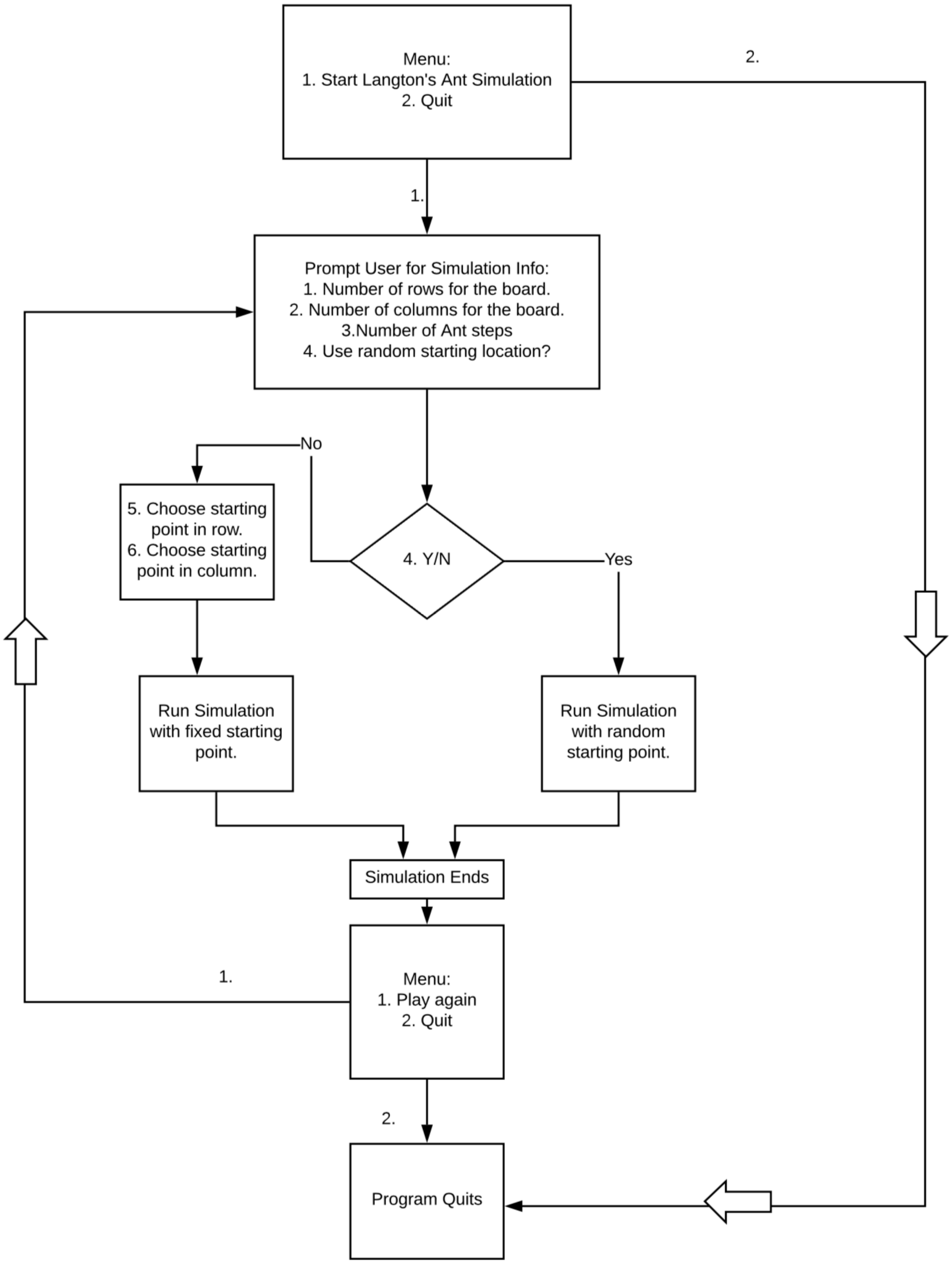
Devin Gendron
Project 1 – Langton's Ant

Flowchart:


Menu:
1. Start Langton's Ant Simulation
2. Quit
2.
1.
Prompt User for Simulation Info:
1. Number of rows for the board.
2. Number of columns for the board.
3.Number of Ant steps
4. Use random starting location?
No
5. Choose starting point in row.
6. Choose starting point in column.
4. Y/N
Yes
Run Simulation with fixed starting point.
Run Simulation with random starting point.
Simulation Ends
1.
Menu:
1. Play again
2. Quit
2.
Program Quits

PseudoCode:

Main()

```
    //start menu function
        //do – while loop
            //while loop
                //ask for and receive input to build simulation
                    /rows/cols/steps/if random/
                //build class object
                //run simulation from object
            //end loop
            //endmenu function
        //end do while
    //simulation quit statement
```

startMenu()

```
    //print statements
    //use inputvalidation func to receive proper return
    //return value
```

endMenu()

```
    //print statements
    //use inputvalidation func to receive proper return
    //return value
```

promptUserMenu()

```
    //using inputval(); code, retrieve rows
    //using inputval(); code, retrieve cols
    //using inputval(); code, retrieve ant steps

    //randomizer()
        //yes or no for random
            //if yes:
                //use randomizer functions to return random rows and cols
            //if no:
                //let user choose:
                //using inputval(); code, retrieve starting x
                //using inputval(); code, retrieve starting y
```

antClass() -take all input from menus

```
    //enum for directions

    //private: holds all values for ant

    //public:
        //constructor

        //getters for all private members

        //setters:
            //set direction setter
            //set ant x loc setter
            //set ant y loc setter

        //simulation function
        //board creation function
        //movement function
        //printboard function
```

simulation()

```
    //dyn 2d array allocation

    //create board set in array

    //loop for ant simulation
        //movefunc
        //printBoard

    //delete 2d array
```

createBoard()

```
    //use nested for loops to set rows and cols
        //set top row
        //set left wall
        //set center
        //set right wall
        //set bottom row
```

moveFunc()

```
    //if direction == ()
        //if != edge case
            //if white tile
                //set dir
                //change tile to black
                //update x & y loc for ant
                //update tile for ant '*'
            //if black tile
                //set dir
                //change tile to black
                //update x & y loc for ant
                //update tile for ant '*'
        //if == edge case
            //180 degree turn
```

printBoard()

```
    //nest for loops to print board
```

setDirection()

```
    //for NSEW
        //If (dir && white tile)
            //turn right
        //if else(dir && black tile)
            //turn left
        //if else (edge case)
            //turn opposite direction
        ……..
```

Test Plan:

startMenu();

Test Case	Expected Outcomes	Observed Outcomes
Input letters	Inputvalidation(); returns “Incorrect entry…”. Restarts loop.	Incorrect entry statement printed, loops to restart input request.
Input symbols or space	Inputvalidation(); returns “Incorrect entry…”. Restarts loop.	Incorrect entry statement printed, loops to restart input request.
Input too low	badEntry bool sets to true, restarts loop until correct input.	badEntry statement printed, loops to restart input request.
Input too high	badEntry bool sets to true, restarts loop until correct input.	badEntry statement printed, loops to restart input request.
Empty input	badEntry bool sets to true, restarts loop until correct input.	badEntry statement printed, loops to restart input request.
Correct Input	Counter for exit increments, print correct entry statement. Program continues.	Returns value to main to continue with program

endMenu();

Test Case	Expected Outcomes	Observed Outcomes
Input letters	Inputvalidation(); returns “Incorrect entry…”. Restarts loop.	Incorrect entry statement printed, loops to restart input request.
Input symbols or space	Inputvalidation(); returns “Incorrect entry…”. Restarts loop.	Incorrect entry statement printed, loops to restart input request.
Input too low	badEntry bool sets to true, restarts loop until correct input.	badEntry statement printed, loops to restart input request.
Input too high	badEntry bool sets to true, restarts loop until correct input.	badEntry statement printed, loops to restart input request.
Empty input	badEntry bool sets to true, restarts loop until correct input.	badEntry statement printed, loops to restart input request.
Correct Input	Counter for exit increments, print correct entry statement. Program continues.	Returns value to main to continue with program

inputValidation();

Test Case	**Expected Outcomes**	**Observed Outcomes**
Input Letters	Incorrect entry counter incremented. Restarts loop.	Restarts loop for proper entry
Input symbols	Incorrect entry counter incremented. Restarts loop.	Restarts loop for proper entry
Input too low	badEntry bool sets to true, restarts loop until correct input.	Restarts loop for proper entry
Input too high	badEntry bool sets to true, restarts loop until correct input.	Restarts loop for proper entry
Input empty string	badEntry bool sets to true, restarts loop until correct input.	Restarts loop for proper entry
Correct entry	Counter for exit increments, print correct entry statement. Program continues.	Returns value to main to continue with program

rowsB();

Test Case	**Expected Outcomes**	**Observed Outcomes**
Input letters	Incorrect entry counter incremented. Restarts loop.	Incorrect entry statement printed, loops to restart input request.
Input symbols or space	Incorrect entry counter incremented. Restarts loop.	Incorrect entry statement printed, loops to restart input request.
Input too low	badEntry bool sets to true, restarts loop until correct input.	badEntry statement printed, loops to restart input request.
Input too high	badEntry bool sets to true, restarts loop until correct input.	badEntry statement printed, loops to restart input request.
Empty input	badEntry bool sets to true, restarts loop until correct input.	badEntry statement printed, loops to restart input request.
Correct Input	Counter for exit increments, print correct entry statement. Program continues.	Returns value to main to continue with program

colsB();

Test Case	Expected Outcomes	Observed Outcomes
Input letters	Incorrect entry counter incremented. Restarts loop.	Incorrect entry statement printed, loops to restart input request.
Input symbols or space	Incorrect entry counter incremented. Restarts loop.	Incorrect entry statement printed, loops to restart input request.
Input too low	badEntry bool sets to true, restarts loop until correct input.	badEntry statement printed, loops to restart input request.
Input too high	badEntry bool sets to true, restarts loop until correct input.	badEntry statement printed, loops to restart input request.
Empty input	badEntry bool sets to true, restarts loop until correct input.	badEntry statement printed, loops to restart input request.
Correct Input	Counter for exit increments, print correct entry statement.	Returns value to main to continue with program

stepsAnt();

Test Case	Expected Outcomes	Observed Outcomes
Input letters	Inputvalidation(); returns "Incorrect entry…"	Incorrect entry statement printed, loops to restart input request.
Input symbols or space	Inputvalidation(); returns "Incorrect entry…"	Incorrect entry statement printed, loops to restart input request.
Input too low	badEntry bool sets to true, restarts loop until correct input.	badEntry statement printed, loops to restart input request.
Input too high	badEntry bool sets to true, restarts loop until correct input.	badEntry statement printed, loops to restart input request.
Empty input	badEntry bool sets to true, restarts loop until correct input.	badEntry statement printed, loops to restart input request.
Correct Input	Counter for exit increments, print correct entry statement.	Returns value to main to continue with program

randomizer();

Test Case	Expected Outcomes	Observed Outcomes
Input letters	Inputvalidation(); returns "Incorrect entry…"	Incorrect entry statement printed, loops to restart input request.
Input symbols or space	Inputvalidation(); returns "Incorrect entry…"	Incorrect entry statement printed, loops to restart input request.
Input too low	badEntry bool sets to true, restarts loop until correct input.	badEntry statement printed, loops to restart input request.
Input too high	badEntry bool sets to true, restarts loop until correct input.	badEntry statement printed, loops to restart input request.
Empty input	badEntry bool sets to true, restarts loop until correct input.	badEntry statement printed, loops to restart input request.
Correct Input	Counter for exit increments, print correct entry statement.	Returns value to main to continue with program

randomX();

Test Case	Expected Outcomes	Observed Outcomes
Receives row number	Creates random row	Random row created. Sets board rows

randomY();

Test Case	Expected Outcomes	Observed Outcomes
Receives column number	Creates random column	Random Column created. Sets board columns.

chooseX();

Test Case	Expected Outcomes	Observed Outcomes
Input letters	returns “Incorrect entry…”. Resets until proper input.	Incorrect entry statement printed, loops to restart input request.
Input symbols or space	returns “Incorrect entry…”. Resets until proper input.	Incorrect entry statement printed, loops to restart input request.
Input too low	badEntry bool sets to true, restarts loop until correct input.	badEntry statement printed, loops to restart input request.
Input too high	badEntry bool sets to true, restarts loop until correct input.	badEntry statement printed, loops to restart input request.
Empty input	badEntry bool sets to true, restarts loop until correct input.	badEntry statement printed, loops to restart input request.
Correct Input	Counter for exit increments, print correct entry statement.	Returns value to main to continue with program

chooseY();

Test Case	Expected Outcomes	Observed Outcomes
Input letters	returns “Incorrect entry…”. Resets until proper input.	Incorrect entry statement printed, loops to restart input request.
Input symbols or space	returns “Incorrect entry…”. Resets until proper input.	Incorrect entry statement printed, loops to restart input request.
Input too low	badEntry bool sets to true, restarts loop until correct input.	badEntry statement printed, loops to restart input request.
Input too high	badEntry bool sets to true, restarts loop until correct input.	badEntry statement printed, loops to restart input request.
Empty input	badEntry bool sets to true, restarts loop until correct input.	badEntry statement printed, loops to restart input request.
Correct Input	Counter for exit increments, print correct entry statement.	Returns value to main to continue with program

simulation();

Test Case	Expected Outcomes	Observed Outcomes
Simulation(rows, columns, steps, xloc, yloc);	Executes move function and print function until loop is over.	Simulates ant movement on the board.

createBoard();

Test Case	Expected Outcomes	Observed Outcomes
createBoard(board, xloc, yloc, direction, steps);	Sets the board and starting location of ant.	Sets board and starting location of ant.

moveFunc();

Test Case	Expected Outcomes	Observed Outcomes
moveFunc(board, xloc, yloc, direction, steps);	Moves ant, updates tiles, and sets the direction.	Moves ant, updates tiles, and sets the direction.

printBoard();

Test Case	Expected Outcomes	Observed Outcomes
printBoard(board, xloc, yloc, direction, steps);	Prints board.	Board is properly printed.

setDirection();

Test Case	Expected Outcomes	Observed Outcomes
setDirection(direction, board, xloc, yloc);	Turns ant in a N, S, E, W direction according to previous direction and tile.	Ant turns in a N, S, E, W direction according to previous direction and tile.

Reflection:

I started Project 1 by reading the assignment requirements and taking notes on the most important parts. Then while at work, I handwrote a list of functions I planned on using and some initial constructions of the Ant Class. With these functions in mind, I then began building a flowchart that would control the flow of the program. At the end of these quick drafts, the logic was sound, and I began fleshing out my functions. Once the rough draft of the logic I wanted to implement in my code was complete, I set up my test table and began writing my source code.

I started with my menus and my main function. I made sure they followed my flowchart and then implemented my input validation function that I created for the previous assignment to validate the responses I wanted. I then tested these menus and functions by my test table and was able receive the proper outcomes. With this part of the program complete, I moved on to the ant class.

The ant class was obviously going to be the most difficult part, but I did extensive brainstorming on paper and was able to sort through a lot of the logic and build a good framework to run the simulation through. I decided I would have my ant object run everything needed to create the Langton's Ant simulation. I had seen how others on piazza or slack had made multiple classes, but I decided to go with my original design and see how far I could get. The ant object received all its members from the menus and had two important functions that ran within my simulation function. The orientation function and the movement function were definitely the most complicated and important parts of my ant class.

With these functions taking up the bulk of the time I spent working on this project, I was able to get a proper board printing with proper input validation checking each menu and entry. I then hit my first wall, I had trouble making my ant take more than one step before getting stuck. I spent hours scouring my code and fixing portions based on TA feedback. During office hours another TA, Ian, suggested a bunch of fixes that could make my code a lot simpler in regard to how my functions were running and some general insight on classes themselves. We then found the main logic issue in my program. I was setting the ant char "*" too soon and then the ant didn't know how to move and would just sit there since I didn't account for the ant properly. When I commented out the lines where I was setting the ant character in my board, my function ran perfectly! All along, it was that simple logic error.

With that behind me, my new task was figuring out how to get the ant character back into the simulation. I slept on it and awoke to the idea of a temporary variable that would hold the tile color for me. So before where I would check for the tile color in my move function, it would check for the ant, and it would then set the tile back from ant “*” to the tile color “ “/”#”. And upon testing it worked perfectly. This was the biggest problem I faced, but once the issue was determined it didn’t take much time to fix.

I learned a lot from this project. It was a good reminder of how to structure and setup my classes, but also how to simplify a lot of code. I constantly found myself trying to do too much and was able to make my code much cleaner after revisions suggested by the TA Ian. I felt comfortable using dynamic memory in this project, because I worked out a lot of my issues with it in Lab 1. If I could do anything differently before starting, I would have studied up on my classes a bit more. I had forgotten some key concepts and had I remembered them, I would have spent less time trying to fix issues that shouldn’t have existed.

Overall, I spent probably too much time in the design phase, however, that meant I had to make less revisions to my overall design. My main issues were logic based and once they were discovered, fixing them was a breeze. I’m really proud of how well my program came out and I’m eager to get started on the next one.