**Carrie Davis**

**Intro Programming 162**

**Project 1 Project Plan**

**Program Requirements/Steps**

Langston Ant Program plan

Create a program that carries out following steps.

1. Requests from user if they want to start program or quit1.
2. Prompts user to pick size of square matrix.
3. Prompts user to pick either to randomly place an ant on matrix or choose where to place.
4. Column and Row location input or randomization are used to place ant on square matrix and orientation is set initially for ant as North or 'N'.
5. Prompts user to pick number of steps or squares the ant will move on board.
6. For each step selected the ant is moved on the board according to Langston's Ant rules.
7. Display results to the user in a specified printed format for each step.
8. Ask user if they want to play again.

```
Menus


main()
Prompt user if start program or quit
input user choice
1 → start program
2 → Do nothing and quit
#%* or != 1 || 2  → while choice is not 1 or 2 prompt user to enter 1 or 2
```

**AntProgram**.run()

Prompt user to pick matrix size between 2 – 20.

*Error check using while until # between 2 – 20 is entered*

Prompt user to enter row and column for ant location on matrix

*Error check using while until # between 1 to matrix size is entered*

*Row:1,column:1 will be first square in matrix representing 0,0.*

Prompt user to enter steps that the ant will take from 1-100.

*Error check using while until # between 1 to matrix size is entered*

Creates antBoard and Ant

While steps are > 0;

Moves Ant on board and follows Langston's Ant rules

Prints board

Counts down one step

Deletes board

**AntBoard** – create a board based on the input from menu
Input matrix size
Creates square dynamic memory bool 2D array that initializes 0 for each sq
uare representing white.

**Ant** – create ant object with location info and orientation.
Input row, column, default orientation = 'N' representing north.
Creates ant with row, column and orientation properties.

**Rules for Langston's Ant on Board**
Perform the following
antOrient = getOrientation(&Ant Object);
antRow = getRow(&Ant Object);
antColumn = getColumn(&Ant Object);
Ant moves differently depending on char for orientation
if (antOrient == 'N')
int row,column = antRow - 1 , antColumn
if board[row][column == 0 && in bounds
board[row][column == 1;
setRow(row);
setColumn(column);
setOrientation('E');

```
            if board[row][column == 1 && in bounds
                    board[row][column == 0;
                    setRow(row);
                    setColumn(column);
                    setOrientation('W');

            //take into account edges turn ant around
            else
                    no change to antRow or antColumn;
                    setOrientation('S');

    if (antOrient == 'E')
            int row,column = antRow, antColumn + 1
            if board[row][column == 0 && in bounds
                    board[row][column == 1;
                    setRow(row);
                    setColumn(column);
                    setOrientation('S');
            if board[row][column == 1 && in bounds
                    board[row][column == 0;
                    setRow(row);
                    setColumn(column);
                    setOrientation('N');

            //take into account edges turn ant around
            else
                    no change to antRow or antColumn;
                    setOrientation('W');

    if (antOrient == 'W')
            int row,column = antRow, antColumn - 1
            if board[row][column == 0 && in bounds
                    board[row][column == 1;
                    setRow(row);
                    setColumn(column);
                    setOrientation('N');
            if board[row][column == 1 && in bounds
                    board[row][column == 0;
                    setRow(row);
                    setColumn(column);
                    setOrientation('S');

            //take into account edges turn ant around
            else
                    no change to antRow or antColumn;
                    setOrientation('E');
```

```
    if (antOrient == 'S')
            int row,column = antRow + 1, antColumn
            if board[row][column == 0 && in bounds
                    board[row][column == 1;
                    setRow(row);
                    setColumn(column);
                    setOrientation('W');
            if board[row][column == 1 && in bounds
                    board[row][column == 0;
                    setRow(row);
                    setColumn(column);
                    setOrientation('E');

            //take into account edges turn ant around
            else
                    no change to antRow or antColumn;
                    setOrientation('N');
```

```
print output location of ant and matrix with black or white squares.

if (print column == antColumn && print row == antRow)

        cout << setw(3) << "*";

else if (ant board[row][column] == 0)

        cout << setw(3) << "  ";

else if (ant board[row][column] == 0)

        cout << setw(3) << "#" ;
```

**Input Variables**

Int choice  // 1 or 2 to start program or quit

int matrixSize //positive integer between 1 - 20

int row //positive integer between 1 - 20 that user will enter or random

int column //positive integer between 1 – 20 that user will enter or random

int steps //positive integer between 1 – 100 that user will enter

**Output Variables**

int antRow //integer represents row on board where ant is located

int antColumn //integer represents column on board where ant is located

int antOrient //direction ant is facing in N,S,E, or W.

**Additional Variables**

int step //counter for steps while loop

int placeAntChoice //1 or 2 to random or user input choice ant row or column

**Testing Plan**

| ***Tests*** | ***Action performed*** | ***Expected output*** |
|---|---|---|
| *Test1:* | Matrix Size Entered 2 | **Matrix Size**: 2 |
| | Row Entered 2 | |
| | Column Entered 0 | Invalid input. Program request new input for row and column because column choice out of board bounds. Then will proceed with new input |
| | Row Entered again as 2 | **Row**: 2 |
| | Column Entered again as 1 | **Column**: 1 |
| | Steps Entered as 2 | **Steps row, column, orient**<br>*note that array starts at 0,0<br>Expect row && col value - 1<br>**Step 0:** 1,0, N |

| | | |
|---|---|---|
| | | **Step 1:** 0,0, E<br>**Step 2:** 0, 1, S<br>**Print Out**<br>`Langston's Ant Simulation`<br>`  *`<br>`Step: 1`<br>`  *`<br>`  #`<br>`Step: 2`<br>`  #    *`<br>`  #` |
| *Test 2:* | Matrix Size Entered 3 | **Matrix size**: 3 |
| | Row Entered 4 | |
| | Column Entered 4 | “Invalid Input.” Program request new input for row and column because row choice out of board bounds. Then will proceed with new input |
| | Row Entered again as 2 | **Row**: 2 |
| | Column Entered again as 1 | **Column**: 1 |
| | Steps Entered as 0 | “Invalid Input.” Program request new input for steps because choice should be between 1 - 100. Then will proceed with new input |
| | Steps Entered again as 10 | **Steps:** 10<br>**Steps row, column, orient**<br>*note that array starts at 0,0<br>Expect row && col value - 1<br>**Step 0:** 1,0, N<br>**Step 1** 0,0, E |

print

**Step 2** 0,1, S

Print

**Step 3** 1,1, W

print

**Step 4** 1,0, S

Print

**Step 5** 2, 0, W

print

**Step 6** 2,0, E Ant skipped step forward and turned around

Print

**Step 7** 2, 1, S

print

**Step 8** 2,1, N Ant skipped step forward and turned around

Print

**Step 9** 1, 1, W

Print

**Step 10** 1, 0, N

**Print Out**

```
 Langston's Ant Simulation

  *

Step: 1
  *
  #

Step: 2
  #    *
  #

Step: 3
  #    #
  #    *

Step: 4
  #    #
  *    #

Step: 5
  #    #
       #
  *
Stopped and turned ant around to avoid going out of bounds.
Step: 6
  #    #
       #
  *
Step: 7
  #    #
       #
  #    *
Stopped and turned ant around to avoid going out of bounds.
Step: 8
  #    #
       #
  #    *
Step: 9
  #    #
       *
  #    #
Step: 10
  #    #
  *
  #    #
```

| | | |
|---|---|---|
| *Test 3:* | Matrix Size Entered @# | Invalid input. Program request new input for matrix because choice is not an integer between 2 and 100. Then will proceed with new input |
| | Matrix Size Entered again as -20 | Invalid input. Program request new input for matrix because choice is not an integer between 2 and 100. Then will proceed with new input |

| | |
|---|---|
| Matrix Size Entered again as 21 | Invalid input. Program request new input for matrix because choice is not an integer between 2 and 100. Then will proceed with new input because |
| Matrix Size Entered again as 20.1 | Program request new input for matrix because choice is not an integer between 2 and 100. Then will proceed with new input |
| Row Entered 1 | **Matrix Size**: 20<br>**Row**: 1 |
| Column Entered 1 | **Column**: 1 |
| Steps Entered as 100 | **Steps:** 100<br>*only showing every 10th step for test abbreviation. Actual program should print all 1-100 steps. |

**Step 0** 0,0, N Ant skipped step forward and turned around

```
Langston's Ant Simulation
 *
```

**Step 1** 0, 0, S

```
Step: 1
  *
```

**Step 10** 0, 1, E

```
Step: 10
  #    *

  #    #
```

**Step 20** 0, 2, N Ant skipped step forward and turned around


**Step 30** 1, 4, W


**Step 40** 1, 4, E


**Step 50** 0, 5, W


**Step 60** 2, 7, W

```
Step: 60
      #     #     #     #     #
  #                                 #
  #     #     #     #     #     #   *
```

**Step 70** 0, 6, S

```
Step: 70
      #     #     #     #     #     *           #
  #                                 #     #     #
  #     #     #     #     #     #     #     #
```

**Step 80** 1, 8, E

```
Step: 80
      #     #     #     #     #     #     #
  #                                       *     #
  #     #     #     #     #     #     #     #     #     #
```

**Step 90** 1,9, N

```
Step: 90
      #     #     #     #     #     #     #     #           #
  #                                             *     #
  #     #     #     #     #     #     #     #     #     #
```

**Step 100** 0,11, E

```
Step: 100
      #     #     #     #     #     #     #     #     #           *
  #                                                   #
  #     #     #     #     #     #     #     #     #     #     #     #
```

**Reflection**

As of late in CS 161 I started implementing incremental development where I would start with a subset of the program and make sure that was working before implementing a new function. I would do one at a time. For this project I built the menus first which through testing and obvious functional need was split into a main() and another file AntProgram with a run() function.

The menus did not call any functions first but just printed out input results and tested then validation by outputting error messages and new requests using while loops. Many of the troubleshooting I accomplished in Lab 1 was easy to apply to the validation steps for this project.

I then created my Ant class and called the set and get methods through the AntProgram menu to make sure they worked, and I could use a pointer to reference the ant object and update the object properties. I had problem with segmentation fault errors because of how I was redefining the array pointer in my class. I had to remove the bool and ** from AntBoard class to fix because it was already defined in my hpp file.

The more challenging area was creating, printing and destroying board created from input matrix size and implementing the Langston's Ant rules. I had to triple check to correct errors in the edge implementation and correctness in the if, if else statements. The code would build or compile correctly and would just send segmentation fault error for certain 2D array code errors or ant placements. Many of these errors had to be resolved through testing and removing code to replace with simple 'cout' statements for testing.

Implemented rules for Langston's ant last as it seemed the most complicated for testing and taking into account all the different cases including edges. For example, the order of which my moveAnt if/else if statements made a big difference. I only discovered this through testing bounds. Then made the statements to check "if" for if the column or row was in bounds and then what the color is for the bool 2D location. Because checking for a value of a row or column not in the bounds of the 2D array caused the Segmentation fault error.

```
if( antOrient == 'S')
    {
        int row = antRow + 1;
        int column = antColumn;

        if((row < boardSize) && (boardPtr[row][column] == 0))
            {
            boardPtr[row][column] = 1;
            (*inAnt).setRow(row);
```

```
            (*inAnt).setColumn(column);
            (*inAnt).setOrientation('W');
            }.
```

I did separate out the if statements based on orientation for the ant. Having the ant first move north always after placed. Then if edge is encountered it will skip moving forward for that step and run around. So, if it's first facing north the first step will be just to turn around and face south or 'S'. I would like to see or have comments on other ways that might have been better to do this. The turnaround and skip step seemed to be easiest to implement.

For testing I did only include some base functional tests in this report. However, I was able to actually perform many more tests and corrections based on those tests. It just seemed an overkill to include or expect for all those tests to be run as base functionality tests. I did discover "edge issues" with test 2 and with test 3 I discovered issue with entering decimal values with "cin" that I used this article to help verify this issue but I used a different solution convert decimal value to integer using static_cast. https://stackoverflow.com/questions/47957584/why-does-ifcin-int-accept-a-decimal-number-in-the-first-iteration-but-no

```
matrixSize = static_cast<int>(inputMatrixSize);
```

If this was an ongoing project, it would be good to note those and create regression tests to include for future testing with updates.