April Castañeda
Professor: Luyao Zhang
CS 162 400 S2019
April 14, 2019

Design + Testing + Reflection

Design

Program goal:

When this program starts, a **menu** appears and prompts the user for two choices. The choices are:
1. "Start Langston's Ant simulation"
2. "Quit."

If the user chooses 2, the program will stop.
If the user chooses 1, the program will go on to a second menu.
If the user enters invalid input, they will be prompted to enter the input again.

A **second menu** appears and the user is prompted for two choices. The choices:
"Would you like to use a random starting location for the ant?"
1. "Y"
2. "N"

If the user enters invalid input, they will be prompted to enter the input again.

A **third menu appears** and the user is then given five/six prompts. They are:
1. The number of rows for the board
2. The number of columns for the board
3. The number of steps during simulation

If the user chose "Y" on the **second menu**, the prompts end here.
Else, two more prompts appear.
5. The starting row of the ant
6. The starting column of the ant

If the user enters invalid input, they will be prompted to enter the input again.

After entering answers to prompts, the program creates a board based on inputted rows and columns and fills the board with white spaces represented by " ". Langston's ant, represented by "*", will then appear on the board. Its starting coordinates will depend on the user's second menu choice. If the user had entered "N", the ant will be placed on the user inputted starting row and column. If the user had entered "Y", the ant will be placed on a random coordinate. The board will print to the screen its initial state and also during every iteration of the ant's steps. The ant will move the user inputted number of steps.
The ant's movements and the value of the space it is on will follow 3 rules:
If the ant is on a white space, it turns right 90 degrees and changes the space to black, or "#".
If the ant is on a black space, it turns left 90 degrees and changes the space to white.
If the ant hits an edge, it will wrap the board around and the ant will appear on the other side.

After the simulation has ended, the first menu should appear again.

April Castañeda
Professor: Luyao Zhang
CS 162 400 S2019
April 14, 2019

Program Organization:

In order to implement the program goal, this program will have seven files total:*
Main.cpp // Main Program
Menu.hpp // Menu Class Specification
Menu.cpp // Menu Class Implementation
Board.hpp // Board Class Specification
Board.cpp // Board Class Implementation
Ant.hpp // Ant Class Specification
Ant.cpp // Ant Class Implementation
(Makefile and Design document not included in the count.)

April Castañeda
Professor: Luyao Zhang
CS 162 400 S2019
April 14, 2019

Program Modularization and Structure:

Main.cpp

Main should include all header files for the three classes: Menu, Board, and Ant.

Main function:

Main program should have a loop in case user wants to play the simulation again.

FIRST MENU:
For first iteration of the main program:
Populate menu1 with prompt script: (“Start Langston’s Ant Simulation”, “Quit”)

For subsequent iterations of the main program:
Populate menu1 with prompt script: (“Play Again?”, “Quit”)

Display menu1 and user enters input.
Menu1 should loop until input is valid.

If user chooses to quit, return 0;
Else, continue program.

SECOND MENU:
Populate menu2 with prompt script: (“Generate random ant coordinates.”, “Enter ant coordinates.”}
Display menu2 and user enters input.
Menu2 should loop until input is valid.

THIRD MENU with SUB-MENUS

Create the first three sub-menus:
Initialize menus(size 1, row/column = 2-100, steps = 1-15000)
Populate menu3a with prompt script: (“The number of rows for the board:”)
Populate menu3b with prompt script: (“The number of columns for the board:”)
Populate menu3c with prompt script: (“The number of steps during simulation”)
Display menu3a-3c and user enters input.
Menu3a-3c should loop until input is valid.
Store input into individual variables - rows, columns, steps.

SUB-MENU - If user wanted to enter own ant coordinates
Create Menu menu3d, 3e objects;
Initialize menus(size 1, max-min = 1-rows/columns)

```
Populate menu3d with prompt script: (“The starting row of the ant:”)
Populate menu3e with prompt script: (“The starting column of the ant:”)
Display menu3d-3e and user enters input.
Menu3e-3e should loop until input is valid.
Store input into individual variables - starting coordinates x, y

If user wanted a random start
Create Ant object; // object is destroyed after random # generation
Generate random x and y;
Store input into individual variables - starting coordinates x, y

Create Ant and Board objects with stored user input.

In loop limited to amount of ant’s steps
Display the Board and Move Ant.

End simulation.

Loop program to prompt user if they’d like to play again.
```

Menu.hpp/Menu.cpp

Private members:

```
int rows;           // how many rows in menu
int minVal, maxVal;     // minimum and maximum values of options in menu
string stringInput;       // user input in string format
int intInput;        // strInput changed to int
string* menuArray; // menu Array
Validate test;     // create Validate class object
```

Public members:

Default Constructor and Non-Default Constructor
Destructor to delete array
Setter functions
Getter Functions
Function that dynamically creates Menu array based on number of rows
Function that displays menu
Function that validates data and converts data from string to integer once validated.

April Castañeda
Professor: Luyao Zhang
CS 162 400 S2019
April 14, 2019

Ant.hpp/Ant.cpp

Private members:

int xCoord, yCoord;
int direction;

Public members:

Default Constructor and Non-Default Constructor
Setter functions
Getter Functions
Function that creates random starting coordinates for ant

Board.hpp/Board.cpp

Private members:

int rows, columns;
int steps;
char boardColor
char ** 2D array for board
Ant object - to access ant information

Public members:

Default Constructor and Non-Default Constructor
Destructor to delete 2D array
Setter functions
Getter Functions
Function that dynamically creates 2d Board based on user input
Function that populates board
Function that displays board
Function that moves ant object within per Langston's ant rules
Function that checks edge cases

April Castañeda
Professor: Luyao Zhang
CS 162 400 S2019
April 14, 2019

Test Cases

#	TEST SUBJECT	INPUT VALUES	EXPECTED OUTCOMES	OBSERVED OUTCOMES
1	First menu	Input < 1 || > 2 || char || string	Menu loops until valid input	Menu loops until valid input
2	First menu	Input = 1	Program continues to second menu	Program continues to second menu
3	First menu	Input = 2	Program exits	Program exits
4	Second menu	Input < 1 || > 2 || char || string	Menu loops until valid input	Menu loops until valid input
5	Second menu	Input = 1	Program continues to only first three questions of third menu. Generates two random numbers.	Program continues to only first three questions of third menu. Generates two random numbers.
6	Second menu	Input = 2	Program continues to all five questions of third menu.	Program continues to all five questions of third menu.
7	Third menu - Sub-menu a	Input < 2 || > 100 || char || string	Menu loops until valid input	Menu loops until valid input
8	Third menu - Sub-menu a	Input > 1 || < 100	Program continues to next sub-menu.	Program continues to next sub-menu.
9	Third menu - Sub-menu b	Input < 2 || > 100 || char || string	Menu loops until valid input	Menu loops until valid input
10	Third menu - Sub-menu b	Input > 1 || < 100	Program continues to next sub-menu.	Program continues to next sub-menu.
11	Third menu - Sub-menu c	Input < 1 || > 15000 || char || string	Menu loops until valid input	Menu loops until valid input
12	Third menu - Sub-menu c	Input > 1 || < 15000	Program continues to next sub-menu.	Program continues to next sub-menu.
13	Third menu - Sub-menu d	Input < 1 || > (row input from menu 3a) || char || string	Menu loops until valid input	Menu loops until valid input
14	Third menu - Sub-menu d	Input > 1 || (row input from menu 3a	Program continues to next sub-menu.	Program continues to next sub-menu.
15	Third menu - Sub-menu e	Input < 1 || > (column input from menu 3b) || char || string	Menu loops until valid input	Menu loops until valid input

#	TEST SUBJECT	INPUT VALUES	EXPECTED OUTCOMES	OBSERVED OUTCOMES
16	Third menu - Sub-menu e	Input > 1 || (column input from menu 3b)	Program continues to next sub-menu.	Program continues to next sub-menu.
17	Simulation Loop	Board and Ant data	Loop to display board and move ant until user inputted number of steps is reached	Loop to display board and move ant until user inputted number of steps is reached
18	Main program loop	End of simulation	Program prompts to play again	Program prompts to play again

April Castañeda
Professor: Luyao Zhang
CS 162 400 S2019
April 14, 2019

Reflection

This project took much longer than I had expected. It probably took me well over 40 hours. In the future, I need to start well ahead of these projects.

The Main Function

I had anticipated that my main file would just include small snippets of function calls to the menu, ant, and board classes. I didn't expect the menu portion of my main function to be as sprawling as it is. I think, in the future, I can package the menus into one or more functions outside of main.

Menus and Sub-menus

The first two menus were easy enough but it was the third menu and its sub-menus that gave me a bit of trouble. The third menu consisted of three or five different questions that required their own set of parameters. In re-thinking it now, perhaps I should've utilized a 2d Array for the third menu so they could all be combined into one object rather than creative five different sub-menus/objects for each prompt.

Input Validation

At first, I had thought to create Input Validation as its own class. I'm still unsure if this would've been the right direction. I instead opted to create an input validation function in the menu class. I had struggled at first on how to validate input if all I wanted were integers. On the first lab, I had played with cin.fail(). I believe I had read either on piazza or slack about taking the input as a string and using stoi(). I thought this was a great idea since strings were arrays. You could then check each array value to validate input. Then, once input was valid, the string could be converted to an integer. After having converted to an integer, you can now compare to maximum and minimum values for the second part of input validation. This was fairly easy to manage once on the right path and implemented intuitively.

Ant Class

I opted to keep the Ant class simple with only its coordinates and direction as its private members. The only involved function in this class is the random coordinate generator. At first I was seeding the rand() function with the time library and quickly found out that my random numbers were all over the map. I opted to use a simpler rand equation that required min and max values from which to generate a number from. This worked out better as the random numbers generated had to exist within the user inputted board dimensions. I also briefly tried putting a Board class member in the Ant Class. I learned quickly that will cause an error when two classes are members of each other (I had made an Ant member in the Board class). I decided in the end to treat the Ant as a board token for the Board class to move around. I concluded that it was best that the Ant did not know the details of the Board class.

Board Class

This was my more complicated class. I spent most of my time re-working this. (Next was the Menu class). I'm not as comfortable with pointers as I probably should be so it took me a while to dynamically create a 2d array and then populate it. In order to create a border around my board, I added two to the number of rows and columns the user had inputted. At first, I didn't store this +2 number into my private variables. Instead, I would allocate the array and populate the array with something like numRows+2. Forgetting parentheses to enclose this formula, forgetting the +2, having to backwards account for it in my counters and if else statements, compounded my difficulty. Then, I also kept getting segmentation faults and memory leak because a confusion like this also trickled down to my destructor. When I finally stored numRows += 2, dynamically allocating an array, implementing population loops, having proper if/else statement became much easier. Also, I found that I was deleting my array in the opposite direction that I should've. This cleared the memory leaks.

Next, was figuring out how to move the ant. At first, I didn't store the initial board color, so once the ant landed on a space, my program couldn't figure out what to change it to after the ant had moved on. So my board would populate with the black character '#' but no ant to be seen. This was solved soon after solving the board color.

Surprisingly, figuring out how to move the ant per Langston's rules and how to maneuver around edge cases was easier than I thought.

Conclusion

This was a difficult project but one that helped me have a better grasp on CS 161 principles. After four days of trying to figure this out, I can't express how much relief and pride I felt in finally seeing that little ant make its path (with no segmentation faults or memory leaks!) In the future, I'd like to improve on simplifying my functions, utilizing modularity, and be more efficient and succinct in my code.