Sangyun Lee

Leesa6@oregonstate.edu

**Requirements for the project1**

1. Created class called 'Ant'
2. Created Two functions for displaying menu and get an input from user
3. Display Board and movement of an ant

**Challenges**

1. Validate user's inputs
2. Create 2d dynamic array based on user inputs
3. Handle the edge case
4. Ask another play

**How I solved those challenges**

1. Since it has to filter non-integer variable, I will use 'cin.fail() and cin.get()'. It will validate all input from users.
2. Dynamic 2d array is built at the time it is called. So, instead using set function, I will create a constructor that accepts row size and column size and other int variables to set up the start position of the ant as parameters.
3. I adopt the one of cases instructor gave, which is ant will go through the edge and come out opposite way. To achieve it, I will use 'if statement'.
   Ex) if (board[x][y+1] == "|" {do something}
4. I will use while loop. Since I can't use break or continue statement, I created the 'reGame variable'. It will get and validate an input from users and returns an int value. So, if the reGame variable holds certain value, then the while loop will be terminated.

**Test Table**

- **Menu() & getInput**

Based on the enum variable, menu(int) set the variable and pass it to getInput, then validate the data and return the required value.

| Test case | Input Value | Driver Functions | Expected Output | Actual Output |
|---|---|---|---|---|
| Wrong Data Type input | 'a' | menu() & getInput() | "Your input wrong, try agaun" | "Your input wrong, try agaun" |
| Filter float type | 2.5 | getInput() while(!validate) | "Your input wrong, try agaun" | "Your input wrong, try agaun" |
| Out of range input | 0 | menu() & getInput() | "Your input wrong, try agaun" | "Your input wrong, try agaun" |

| | | | | |
|---|---|---|---|---|
| Correct input | 2 | menu() & getInput() | “Thank you for using this program”, close | “Thank you for using this program”, close |
| Correct input | 1 | menu() & getInput() | random location on boadrd | random location on boadrd |
| Correct input | 2 | menu() & getInput() | “Row position of ant…..” | “Row position of ant…..” |
| Correct input | 1 | menu() & getInput() | New game start, “Row size of…..” | New game start, “Row size of…..” |

Ant.CPP test as one

- antSimulation() : constructor

Since, the constructor accepts four int variables to set the board and ant location, I tested. And getInput function validates all the user’s input before entering it in constructor

| Test case | Input Value | Driver Functions | Expected Output | Actual Output |
|---|---|---|---|---|
| Set constructor With 4 variables | 5,5,2,2 | antSimulation () | 5x5 board / ant on [2,2] | 5x5 board / ant on [2,2] |
| Set constructor With 4 variables | 10, 8, 4, 5 | antSimulation () | 10x8 board / ant on [4, 5] | 10x8 board / ant on [4, 5] |
| Set constructor With 4 variables | 10, 10, 10 ,10 | menu() & getInput() | 10x10 board / ant on [10, 10] | 10x10 board / ant on [10, 10] |

- setStep() :

| Test case | Input Value | Driver Functions | Expected Output | Actual Output |
|---|---|---|---|---|
| Set number of steps for ant | 10 | setStep() step = ; | Remaining step : 10 to 0 | Remaining step : 10 to 0 |
| Set number of steps for ant | 2 | setStep() step = ; | Remaining step : 2 to 0 | Remaining step : 150 to 0 |
| Set number of steps for ant | 15 | menu() & setStep() step = ; () | Remaining step : 30 to 0 | Remaining step : 30 to 0 |

- playGame() :

| Test case | Input Value | Driver Functions | Expected Output | Actual Output |
|---|---|---|---|---|
| Checking ant movement | n/a | playGame() : | 5x5 Board End at [3][3] | 5x5 Board End at [3][3] |
| Set number of steps for ant | n/a | playGame() : | 10x8 Board End at [3][6] | 10x8 Board End at [3][6] |
| Set number of steps for ant | n/a | playGame() : | 10x10 Board End at [1][10] | 10x10 Board End at [1][10] |

**Reflection**

Program design and test table documents is very useful. When someone created programs, especially for beginner, it is easy to skip test in different ways. This is because they haven't built complex program before, or they are not just used to test it. I was one of them. This ant programing was more complex than any assignments from CS161. So, by repeating the test case until I get the result that I expect to have, I was able to find logical errors. Direction of the ant based on the board color and changing the board color were hard.

Therefore, when I was testing the ant simulation, I tested 3 ways, left top corner, middle and right bottom corner so that I can see how the ant react when it faces the edge. It helped me figure out the logical errors and revised code. Throughout the test case, even I realized the importance of how to set up test case. Programmer also think about how to test their program rather than just try random cases.

And, it was great practice to learn more about 2d dynamic array. It was confusing to use pointer of pointer before, but while programing this project, I learned a lot through research, and questions and answers on piazza.