**Derek Yang**

**Document (Design + Reflection)**

Program Design

Problem: Create an implementation of Langston’s Ant. The ant will move across a grid according to the following set of rules:

* If the ant is on a white space, it will turn right 90 degrees and change the space to black
* If the ant is on a black space, it will turn left 90 degrees and change the space to white

The ant will then move forward to the next space. The ant must keep moving until it reaches the total number of moves.

Ant Class

* The ant class will contain all the information
  + There will be get and set functions used in the ant class to get and set variables for tracking the ant
* The following will be tracked: the board, the ant’s location, and the orientation of the ant. The orientation of the ant will be divided into direction and left/right
* The ant will be notated by a \*, blank square by a ‘ ‘char and ‘#’ for a black square.
* Each variable listed above will have it’s own get and set function. There will probably be a need for a function to track the total number of steps and the current number of steps.

Main

* The main will contain the menu function and prints the board
* The board will need to be a double pointer because it is a 2d dynamic array
* The user will be asked if they would like to choose to start the simulation, start a simulation with a random starting location, or quit.
  + If the user chooses to start the location it will ask the following of the user:
    - Grid width, grid height, ant starting location, and number of steps. The random start location will not ask for the start location from the user.

Board

* There will be a board class that keeps all the board moves. The ant class keeps the information of board position but the board class will contain the functions to move around the ant
  + Here are the anticipated functions
    - Turn square color and change left/right
      * If current square is black, set LEFT and WHITE
      * If current square is white, set RIGHT and BLACK
    - Change direction upon hitting a square depending on color
      * If current square is black, set LEFT
      * If current square is white, set RIGHT
    - Track the total number of steps and see if it has reached max
      * If the ant is facing a certain direction, set new direction to opposite
    - See if the ant has reached the edge, if so turn the ant around and continue
      * If the ant future position exceeds max, then turn the ant around the opposite direction
    - Delete the board
      * Delete the first level array and then delete each point to array within the array

Input Validation

* The user will be asked mainly integers in the menu options. The cases are usually within a range of answers. Therefore, I will create a function that checks for an int, and then if that int is within a specified range.
  + Ex: choose 1 to 3 will ask for an int within range 1 to 3
  + Ex2: choose starting grid width will be min 1 and max int

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case** | **Input Values** | **Functions** | **Expected Outcomes** | **Observed Outcomes** |
| Input too low | Input is less than the min | Main()  Validation()  If input < min | Loop back and ask the user to re-input | Continually asks user to enter a higher int |
| Input too high | Input is more than the max | Main()  Validation()  If input > max | Loop back and ask the user to re-input | Continually asks user to enter a lower int |
| Input in correct range | Input is within max and min | Main()  Validation()  If min>input>max | Accepts the correct answer and exits the while loop | Loop stops and accepts correct int |
| Input is not an int | Input is a string, bool, float or other variable | Main()  Validation()  If input != int | Loop back and ask the user to re-input an int | Loops until an int is entered within the designated range. |
| Ant hits a corner | No input | antEdge()  if antPosition>max then turn the ant around depending on current direction. This may repeat because it may hit another edge immediately after | Ant turns around the opposite direction which it came from | Ant will hit the corner and continues to hit another corner until it is completely turned around |
| Ant hits the edge | No input | antEdge()  if antPosition>max then turn the ant around depending on current direction. | Ant turns around the opposite direction which it came from | Ant gets turned around. It registered that it has reached the end of the array length and is turned the opposite direction. |
| Ant starts going over 10,000 steps | No Input | Ant()  Grid()  Ant starts to create a highway after 10,000+ steps | Ant starts to create a higihway after 10,000+ steps | At a certain point, the ant starts to create a highway |

Reflection

Making the planning document and then actually programming the langston’s ant implementation was different than what I expected. I should have listed out all the expected variables in each file first in my plan. Having a plan definitely helped me in the programming process. I ended up using the “current” variables to keep track of where the ant would be rather than where it currently was. Creating a bunch of get and set functions really helped me to keep track of the ant. Deleting the array was harder than I had originally thought. I ended up having to do a bunch of research to delete a dynamic array.

In the following paragraph, I am going to list the changes I made from the plan and how I solved the problems. I ended up setting the min and max validation function as a single function instead of separating it into min and max. This allowed me to re-use my validation function and I plan on using it in future projects. Another change I made was that I had to keep track of my directions instead of just setting a direction. I used enums instead of my original plan of using strings. I felt that using enums was a cleaner way and helped with the readability of the project. Turning around the ant was a huge project time killer for me. I had to list all the cases in which an ant would have to turn around. I think in the future, I would like to add a function test plan so that way I have a good plan of my functions as well. It was hard to come up with each case and similar cases to combine for north, east, west, south setting directions. I also ended up only adding to the ant’s current position if it exceeded max. Originally I was planning to just subtract but found it was easier to just not add if the ant exceeded max. I also did not account for how to keep a placeholder for when the ant hit the edge. In this case, I needed to come up with a way to keep the current color, ant position, and the future case.

In the following paragraph, I will discuss what I learned from this project. I found that it was better to list the variables or at least all the variables before actually starting. I may even start to add in my header files into my project plan. Listing my sources as comments also helped me to figure out certain concepts, which I could look back on later. I also found that in most cases, a shallow description of the function did not help me at all with implementation. I should plan the function with greater detail in the future and write down which other components of the program are effected by the function. I also found that I need to be consistent with how I declare variables and combine them. In some cases, I would get garbage values and I found that it was because I was not diligent in using my variable types. In the future, I would also like to look at ways to make my code more efficient so that programs such as this one run faster and take up fewer resources.