Andrew Wilson (wilsoan6)
CS 162-400 Winter 2019

Langdon's Ant

Program Flow

Present main menu to user
    If user selects 2, quit immediately.
    If user selects 1, proceed to series of prompts regarding board size
    (If user selects anything other than 1 or 2, reprompt user)

Prompt user if they want a random starting space

Prompt for number of rows (between 1 and ???)
Prompt for number of columns (between 1 and ???)

If nonrandom starting space
    Prompt for starting row (between 1 and user input row)
    Prompt for starting column (between 1 and user input row)
If random starting space
    Assign random starting space

Prompt for number of steps (between 1 and ????)
(Prompt for stating direction?)

Instantiate the board
Place ant in starting location

Run ant movement algorithm one step
        If ant is against boundary, teleport to other side
Print the board
Repeat x user entered number of steps

Prompt user to play again or quit

Andrew Wilson (wilsoan6)
CS 162-400 Winter 2019

Ant movement algorithm

- If on white
    - ✧ if facing NORTH
        - now facing EAST
        - move one step EAST
        - check bounds
            - ~~➢ if against wall, face SOUTH~~
            - ~~➢ if still against wall, face WEST~~
                - ~~✧ infinite loop possibility?~~
            - ➢ if out of bounds, teleport to opposite side of board
        - previous tile is black
        - current tile is *
    - ✧ if facing EAST
        - now facing SOUTH
        - move one step SOUTH
        - check bounds
            - ➢ if out of bounds, teleport to opposite side of board
        - previous tile is black
        - current tile is *
    - ✧ if facing SOUTH
        - now facing WEST
        - move one step WEST
        - check bounds
            - ➢ if out of bounds, teleport to opposite side of board
        - previous tile is black
        - current tile is *
    - ✧ if facing WEST
        - now facing NORTH
        - move one step NORTH
        - check bounds
            - ➢ if out of bounds, teleport to opposite side of board
        - previous tile is black
        - current tile is *
- If on black

  (Same as above, but flip cardinal directions and black/white)

Andrew Wilson (wilsoan6)
CS 162-400 Winter 2019

Test Plan

Test case	Input Values	Affected functions	Expected outcomes	Observed outcomes
Negative input	Input < 0	main() gameboard setup functions	Reprompt user for positive input	Reprompt user for positive input
Input is 0	Input == 0	main() gameboard setup functions	Reprompt user for positive input	Reprompt user for positive input
Input is too high	Steps > 15000 Rows/cols > 100	main() gameboard setup functions	Reprompt user for smaller input	Reprompt user for smaller input
User enters float	Input = "1.1"	main() gameboard setup functions	Reprompt user for correct input	Reprompt user for correct input
User enters letters after numbers	Input = "1a"	main() gameboard setup functions	Reprompt user for correct input	Accepted the integer, but cleared the letters from the buffer; fixed by updating validation to check to for letters
User enters spaces between numbers	Input = "1 1"	main() gameboard setup functions	Reprompt user for correct input	Accepted the first integer but cleared the remaining integers from the buffer; fixed by updating validation to check for spaces
Ant rotates clockwise on white	-	antMove()	Ant rotates clockwise and advances	Ant rotates clockwise and advances
Ant rotates anticlockwise on white	-	antMove()	Ant rotates anticlockwise and advances	Ant rotates anticlockwise and advances
Ant hits edge	-	boundsCheck()	Ant teleports to opposite side	Ant overwrote edge boundaries; fixed by adjusting bounds offset by 1
Ant forms highway	-	antMove()	Ant correctly forms highway at ~12000 steps	Ant formed small highway at 500 steps; fixed by correcting a SOUTH to NORTH

Andrew Wilson (wilsoan6)
CS 162-400 Winter 2019

Reflection

I spent a substantial amount of time gaming the ant movement algorithm and planning the main menu (I joined the OSU hackathon over the winter break and my team's program was a simple movie database with carts and submenus for different types of search, so the menu for this program was relatively straightforward), so I did not need to make many large changes from my initial design with the exception of how to handle the ant hitting the edge of the board: I initially planned to rotate the ant again before moving it forward. However, when I sat down and sketched out the algorithm it quickly became convoluted and seemed difficult to implement, so I saved myself a headache and simply teleported it to the other side.

The other noteworthy problems I encountered were with input validation (as noted in the test plan) and with printing the board. I had written some basic input validation functions for the abovementioned hackathon project but they did not handle integers followed by spaces and letters, so I needed to add code to handle each case. (When I have the bandwidth I intend to completely rewrite the functions to read input into a string as suggested.)

I had an issue printing the board: it seemed to be instantiating and filled with white spaces correctly in the constructor, but only printed garbage when I tried to access it with a class function. After an embarrassingly long time troubleshooting, I realized I instantiated a completely new array in the constructor rather than the one that was part of the class ("char** gameboard =" instead of just "gameboard =").

From this project, I learned to make sure I'm accessing class members rather than completely separate variables I happened to give the same name to, I've improved my understanding of input validation, and I've learned to make sure to follows the specs and not make extra work for myself for no good reason.