Kyle Karthauser
Project 1 CS162 Spring 2018

Project 1: Langton's Ant

PRIOR TO PROGRAMMING, this was my sketched out design:

ANT CLASS

- Board object
- Position
    - Grid coords
    - White or black square
    - Facing
- Evaluate
    - What is the turn count?
        - IF REACHED SIMLIMIT:
            - End
        - IF NOT SIMILIT:
            - simCount++
    - Is move valid?
        - IF +/-1 = EDGE OF BOARD: No.
            - Flip 180*; go back to beginning of evaluate()
        - IF NOT EDGE: Yes.
            - Call Move() function
- Move
    - Facing direction + 1
        - IF FACING EAST: +1 y-dimension
        - IF FACING NORTH: -1 x-dimension
        - IF FACING SOUTH: +1 x-dimension
        - IF FACING WEST: -1 y-dimension
        -
- React
    - If on a **WHITE** square:
        - 1) Turn **right** 90 degrees
        - 2) Change square to **BLACK**
    - If on a **BLACK** square:
        - 1) Turn **left** 90 degrees
        - 2) Change square to **WHITE**

MENU

- Extra Credit
    - Ask if the user would like a RANDOM starting point for the ant. This prompt should be given prior to asking the user to specify a starting point
- 1. Start Langton's Ant simulation
    - Number of rows?
    - Number of columns?
    - Number of steps in simulation?

- Random starting point?
  - If YES: Random starting point. Begin simulation.
  - If NO
    - Starting row of ant?
    - Starting column of ant?
- 2. Quit

MENU DESIGN

- Do-while loop
  - Do...
  - While runAgain == true
    - Default to false. After simulation ends, we reach prompt to “Run again?”

Test Case	Input Values	Driver Function	Expected Outcome	Observed Outcome
User asks for 5x5 grid	X = 5, Y = 5	Board::initBoard();	Dynamic 5x5 2D array is created	Dynamic 5x5 2D array is created
User asks for 3x8 grid	X = 3, Y = 8	Board::initBoard();	Dynamic 3x8 array is created	Segmentation fault
User asks for 8x3 grid	X = 8, Y = 3	Board::initBoard();	Dynamic 8x3 array created	A-Ok
The ant hits an edge	Case specific	Ant::evaluate(); Ant::move();	Ant flips the opposite direction and moves	Ant flips the opposite direction and moves
Langton’s first move	User sets up initial location	Ant::orient(), Ant::evaluate(), Ant::Move()	Ant moves the direction its facing	Ant rotates right due to white space before moving
Ant starts on a black square	Case specific	Ant::rotate(string);	Ant rotates right 90*	Ant rotates right 90*
Ant starts on a white square	Case specific	Ant::rotate(string);	Ant rotates left 90*	Ant rotates left 90*

DESIGN AFTER PROGRAMMING

1. I ditched the idea of a Menu class. I think it’s beyond me, but the ideal would be to make an object which would allow me to easily configure prompts and options and potential return values.
2. Ant class contains a Board object. Default constructor for the Ant class sets up all the dominoes:
   a. Prompt user for number of iterations in simulation
   b. Call Board::initBoard(); to set up the Board object included with the Ant object

c. initBoard() initializes the pointer to a pointer of chars that is contained within the Board object according to user input
d. We return to the Ant() constructor, where the user can choose a random starting and facing position for Langton the ant, or can specify all of this themselves. Randomization is handled in the constructor; custom setup is handled by the Ant::orient() function.
e. Ant::reOrient(enum) function is used to change the direction which Langton faces. Use of an enum makes it much easier to orient yourself to what is happening.
f. Then we get to Ant::evaluate(). Here is its process:
    i. Are we at the final iteration of the simulation? If not, continue. Otherwise, end.
    ii. Is it the first turn? If so, when Langton moves, move in that direction (unless he hits an edge).
    iii. If this isn't the first turn, Langton considers the color of the tile he's on before moving. He must Ant::rotate() according to the color before moving.
    iv. After rotating, we switch the color of the tile.
    v. Check for edges, see if a flip 180* is necessary.
    vi. If there's no edge, move.
g. The Ant::move() function updates a private array "int position[2]." This is used for calculating moves but also for rendering the board as we progress. It occured to me while programming that I wouldn't be able to hold two values in one cell in the board array--that is, a square could be white, black, or have Langton, but I wasn't able to hold two out of those three at once. So my solution was to have the Board::dispBoard() function "overlay" Langton's * symbol over a square but never be written into the array. The array was reserved for black/white square representations.
h. The "elegant" aspect of this design is that I can initialize the ant object, initialize the Board array at the same time, then simply loop the Ant::evaluate() function (which returns a boolean value in order to control a do-while loop in main) to run the simulation. And when we end and the Ant object goes out of scope, the Board object does, too, and its destructor is called implicitly taking care of any memory leaks.

- Surprises
    - After 3 days of working on the assignment, I realized that I had my X and Y coordinates mixed up. This threw me for quite a loop as I tried to figure out why in the hell east/west/north/south were causing completely unpredictable behavior. I should know by now that each and every time you add a function, especially such a fundamental function. It's so tedious and oftentimes complicated to re-engineer your main function to test this, or that, or the other thing. Lesson learned for the tenth time: test every function as you add it, and test it by itself whenever possible before mixing it in with others!
    - Using an enum as a return type requires this syntax: "object_class::enum_name object_class::function()". But if you want to use an enum as a function parameter, it's a lot more straightforward: "void function(enum_class variable_name)."
    - Std::cin >> cannot write to an enum variable.

- I chose to separate the Board object from the Ant object. This may be helpful in the future if I ever use the Board class again (it's essentially built around a dynamic 2D array), but for this, I'm not exactly sure if it was constructive or not. On the one hand, it's nice to have a header file to reference indicating exactly which functions are compatible with which objects. On the other, it's another pair of files I have to maintain and reconcile.