Ryan Ellis
Student #: 933047412
CS162

Langston's Ant (Project 1) Design Document

Finding the Classes:

Problem Domain: Per Wikipedia, "**Langton's ant** is a two-dimensional Turing machine with a very simple set of rules but complex emergent behavior." The machine consists of an ant that moves across a board composed of cells that can alternate colors (black and white). The ant follows a simple set of rules while traversing the board (see below).

If the ant enters a white cell it turns 90 deg clockwise, changes the square to black, and moves to the next cell.

If the ant enters a black cell it turns 90 deg counter-clockwise, changes the square white, and moves to the next cell.

The board displays the current state as the ant travels. The size of the board (rows and columns) is defined by the program user.

List of Nouns:

- ant
- board
- cells
- colors
- current state
- size of board (rows and columns)

Class Responsibilities:

Ant:

- knowing where it is on the board

- which way it is facing

- color of the cell it is in

Board:

- knowing current state of the board; which cell is which color (black, white, ant)

- how big the board is (rows and columns)

Code Plan/Notes:

General Requirements:

- Create 2D array base of the # of rows/columns specified by the user.

Ryan Ellis
Student #: 933047412
CS162

- Run simulation for user specified # of steps.
- Ant class to keep track of # of steps, direction ant is facing, color of cell occupied by the ant.
- Board class that keeps track of the current board.
- Validate all the inputs.

Basic Sketch and Notes on Menu F(x):

- Inputs (variables needed):
    - # of rows
    - # of columns
    - # of steps
    - Ant start point
        - Break this down into row and column variables.
- Original Design:
    - Had menu with 5 options.
        - 1. Set # of board rows.
        - 2. Set # of board columns.
        - 3. Set ant's start position.
            - Sub-menu for row and column
        - 4. Set # of steps for ant to take.
        - 5. Run command.
    - This plan was scrapped after design/completion of Board and Ant classes. I was not sure how to ensure that all parameters needed for the Board and Ant objects would be set when user entered command to run simulation.
        - Changed to menu with 3 options.
            - 1. User specifies all parameters.
                - Option runs prompt that requires user to enter all parameters. User cannot continue until all variables have acceptable values in them.
                - These are then passed to the Board and Ant objects.
            - 2. Random ant start point.
                - Options runs prompt. Gets Board size and number of steps

from user. Calculates random coordinates based of the size of the board.

- Passes these values to Board and Ant objects.

- 3. Quit.
  - Contains only a break statement.

- Test plans for input validation implemented for getting user input in the menu and setting the simulation parameters (size of board, ant's starting point, number of steps) have been included in this packet (ref. MenuTestPlan.pdf and ParameterTestPlan.pdf).

Basic Sketch and Notes on Board Class

- Functions:
  - Board::Board(rows, columns)
    - Sets the boardRows and boardColumns member variables
    - Also, sets member variable for the pointer to the 2D array equal to NULL.
  - Board::~Board
    - Destructor that frees up the memory allocated to the 2D array.
  - createBoard()
    - Takes the boardRows, boardColumns and board pointer, and creates a 2D array.
    - Then, loads the array with '#' characters (symbolizes a black cell).
  - boardState()
    - I was not sure if I would need this or not. The idea was a function that would interact with the Ant class. Ex. Update board based off ant's location, change color of cells appropriately as ant moved across the board. I ended up dividing this into two different functions.
      - antOnBoard(row, column)
        - Changes the cell that is the ant's current position to the '*' symbol (symbolizes the ant).
      - previousPointColor(row, column, cell's color)
        - Changes the cell the ant was in to the correct color.
  - getRows and getColumns
    - Used by the Ant object to see whether it was about to move off the board.
  - printBoard()
    - Realized after I started creating the Ant class, I would need a function to

get the Board object used by the Ant object to print itself out.

- Variables:
    - board[][]: Holds the 2D array.
    - boardRows and boardColumns: Hold the size of the board.

Basic Sketch and Notes for the Ant Class

- Original Idea for How to Move the Ant:
    - Ant position
        - Ex. currentPosition = board[y][x]
    - Facing direction
        - Ex. facingPosition = board[y-1][x]
            - Note: This would be north initially.
    - Make current position = facing position
        - Ex. currentPosition = facingPosition
    - Figure out new facing direction
        - Ex. If facing North currently
            - If currentPosition = white
                - Then clockwise rotation
                    - Ex. facingPosition = [y][x+1]
            - If currentPosition = black
                - Then counter-clockwise rotation
                    - Ex. facingPosition = [y][x-1]
    - This changed over time. Major revisions included the below.
        - As mentioned above, I realized I would need to add in functionality to see if the Ant object's move would put it off the board.
            - If so, I decided to the ant would move over to the opposite side of the board.
        - Instead of using another coordinate to keep track of the direction the Ant object was facing, I decided to use a structure made up of the cardinal directions (North, South, East, West). This was a little more readable to me.
            - The Ant object's movement would determined by sending the current facing direction into a switch statement that would move the current position.

Ryan Ellis
Student #: 933047412
CS162

- Functions:
    - Ant::Ant(pointer to a board object, number of steps, start row, start column)
        - Constructor that initializes all the member variables
    - Ant::~Ant()
        - Destructor that sets the pointer to the board object back to 0.
    - move()
        - Function that holds all the code for moving the ant. Based off the algorithm I described earlier.
        - Main structure is a big for loop that runs for however many steps are specified.
            - Inside that for loop is a switch statement that determines the ant's point based off its current heading. Includes if statements to deal with the ant moving off the board. Calls the turn() and updateBoard() functions.
    - updateBoard()
        - Passes all the data needed to keep the Board object up to date as the ant moves around.
    - turn()
        - Changes the ants heading based off the ants previous heading and the color of the square the ant has moved into.
- Variables:
    - currentRow and currentColumn: Holds the Ant objects current position.
    - previousRow and previousColumn: Holds the previous position of the ant. Is flowed back to the Board object through the boardUpdate() function to ensure it is updated properly.
    - currentPointColor: Used by the move() and turn() function to help move the Ant object correctly.
    - previousPointColor: Used to flow the previous point color back to the Board object, so it can be updated properly.
    - currentHeading: Holds the structure value corresponding to the Ant object's current heading.
    - steps: Holds the number of steps the Ant object is specified to take.