Project 2 Reflection

Byron Kooima

CS/162

July 09, 2017

Prof. Luyao Zhang

Project 2 Reflection

Project Description

The main focus of this program is to implement the Langton's Ant scenario. The board the Ant moves around on is considered an array (2D) of cells. Each movement/step the Ant takes will change the value of the array cell and the movements are based on a specific set of rules. The basic rules of the Ant scenario are:

1. In a blank (' ') square, turn right 90 degrees and change the square to '#'
2. In a '#' square, turn left 90 degrees and change the square to blank (' ')

The Ant will be represented by the '@' sign and the program will allow the user to select the scenario options through the use of a menu. The following options will be made available to the user:

1. Set the table size based on user input for the height and width of the board.
2. Set the Ant's starting position constrained by the height and width of the board.
3. Set the Ant's starting direction based on North/East/South/West coordinates.
4. Set the Ant's number of moves until the simulation ends.

The main menu option will be to start or continue the simulation using a default set of values. If any changes are made using the other menu options, the simulation will run with the values based on the user input.

Another aspect of the simulation is how the Ant will handle moving across the edge of the board. The implementation for this program will have the Ant wrap around to the opposite side of the board and continue on with the simulation.

Project Design

Psuedo Code

Main.cpp

```
Declare pointer to ant as New Ant
Declare pointer to board as New Board
Declare menu as userMenu
Call menu.makeSelection()
WHILE(menu is not equal to 8)
  IF menu is equal to 1
    Set cntMoves to 0
    Call board.setAnt(pointer to ant)
    Set numMoves = Call ant.getMoves()
    WHILE(cntMoves is less than numMoves)
      Call board.output()
      Call board.moveAnt()
      Sleep function for 250 milliseconds
      Increment cntMoves
      Print the number of cntMoves
    END WHILE

  ELSE IF menu is equal to 2
    Print “enter the height of the Langton Ant board”
    Set verString equal to “Board size must be less than 80”
    Call inputValidation.SafeInput (data type, user input for height, verString)
    Print “enter the width of the Langton Ant board”
    Call inputValidation SafeInput (data type, user input for width, verString)

    Call board.setBoard(height, width)
    Call board.setHeight(height)
    Call ant.setY(board.getHeight() divided by 2)
    Call board.setWidth(width)
    Call ant.setX (board.getWidth() divided by 2)

  ELSE IF menu is equal to 3
    Set curHeight = Call board.getHeight()
    Print “enter the height for the ant’s starting position”
    Print “Must be less than board height” + curHeight
    Set verString equal to “Ant height must be less than board height”
    Call inputValidation SafeInput (data type, user input for ant height, verString)
    Call ant.setY(ant height minus 1)

    Set curWidth = Call board.getWidth()
    Print “enter the width for the ant’s starting position”
    Print “Must be less than board width” + curWidth
    Set verString equal to “Ant width must be less than board width”
    Call inputValidation SafeInput (data type, user input for ant width, verString)
    Call ant.setX (ant width minus 1)

  ELSE IF menu is equal to 4
    Call ant.setX(Random number between 0 and board width)
    Call ant.setY(Random number between 0 and board height)
    Print “The random position is: “ + ant height + ant width

  ELSE IF menu is equal to 5
    Print “Enter ant’s starting direction”
    Set verString equal to “0 for North, 1 for South, 2 for West, 3 for East”
    Call inputValidation SafeInput (data type, user input for ant direction, verString)
    Call ant.setDir(ant direction)

  ELSE IF menu is equal to 6
    Print “Enter ant’s number of moves”
    Set verString equal to “Number of moves must be less than 1000”
    Call inputValidation SafeInput (data type, user input for moves, verString)
    Call ant.setMoves(ant’s number of moves)

  ELSE IF menu is equal to 7
    Delete the current ant object
    Delete the current board object
    Declare ant equal to new Ant
    Declare board equal to new Board
  END IF

  Call menu.makeSelection()
Delete the ant object
Delete the board object
RETURN 0
```

End main.cpp

Class userMenu

```
  Print “Please select an option from the menu”
  Set verString equal to “menu options”
  “1. Begin/Continue simulation”
  “2. Set table size”
  “3. Set ant’s starting position”
  “4. Set random starting position”
  “5. Set ant’s starting direction”
  “6. Set number of moves for the ant”
  “7. Re-initialize the Langton’s Ant scenario”
  “8. Exit program”
  Print verString
  Call inputValidation SafeInput (data type, user input for menu selection, verString)
  RETURN user’s menu choice
End Class

Class langtonAnt
  Declare enumerated Direction (DIR_NORTH, DIR_SOUTH, DIR_WEST, DIR_EAST)
  Declare a_direction as Direction
  Contructor for Ant()
    Set a_direction to DIR_NORTH
    Call setMoves(80)
    Call setX(40)
    Call setY(20)
  Subprogram getX()
    Return ant’s x position
  Subprogram getY()
    Return ant’s y position
  Subprogram getDir()
    Return a_direction
  Subprogram getMoves()
    Return ant’s number of moves
  Subprogram setX(x)
    Set ant’s x position to x
  Subprogram setY(y)
    Set ant’s y position to y
  Subprogram setDir(d)
    Set ant’s direction to d
  Subprogram setMoves(z)
    Set ant’s number of moves to z
  Subprogram turnL()
    Case: Ant’s direction is DIR_NORTH. Change direction to DIR_WEST
    Case: Ant’s direction is DIR_WEST. Change direction to DIR_SOUTH
    Case: Ant’s direction is DIR_SOUTH. Change direction to DIR_EAST
    Case: Ant’s direction is DIR_EAST. Change direction to DIR_NORTH
    Default: Ant’s direction is DIR_NORTH
  Subprogram turnR()
    Case: Ant’s direction is DIR_NORTH. Change direction to DIR_EAST
    Case: Ant’s direction is DIR_EAST. Change direction to DIR_SOUTH
    Case: Ant’s direction is DIR_SOUTH. Change direction to DIR_WEST
    Case: Ant’s direction is DIR_WEST. Change direction to DIR_NORTH
End Class

Class langtonBoard
  Declare ON equal to 1
  Declare OFF equal to 0
  Declare pointer to a_ant as Ant
```

```
Constructor for Board()
  Call setHeight(40)
  Call setWidth(80)
  Declare a_board as new pointer to array of size Board height
  FOR each row on board
    Declare a_board[mRow] equal to new column of size Board width
  FOR each row on a_board
    FOR each column on a_board
      Set a_board cell to 0

Destructor for Board()
  FOR each row on a_board
    Delete each column
  Delete a_board
  Set a_board to 0

Subprogram setAnt(pointer to ant)
  Set a_ant to ant

Subprogram flipColor(pointer to ant)
  Set color to a_board cell at current ant position
  IF color is equal to OFF
    Switch a_board cell at current ant position to ON
  ELSE
    Switch a_board cell at current ant position to OFF

Subprogram getColorAt(pointer to ant)
  Return a_board cell color at current ant position

Subprogram setHeight (x)
  Set HEIGHT to x

Subprogram setWidth (y)
  Set WIDTH to y

Subprogram getWidth()
  Return WIDTH

Subprogram getHeight()
  Return HEIGHT

Subprogram setBoard(h, w)
  FOR each row on a_board
    Delete each column
  Delete a_board
  Set a_board to 0
  FOR each row on board
    Declare a_board[mRow] equal to new column of size Board width
  FOR each row on a_board
    FOR each column on a_board
      Set a_board cell to 0

Subprogram moveAnt()
  Call flipColor(a_ant)
  IF color of a_ant is OFF
    a_ant.turnL()
  ELSE IF color of a_ant is ON
    a_ant.turnR()
  Call checkBounds()

Subprogram checkBounds()
  IF a_ant x position is less than 0
    Set a_ant x position to board width – 1
  IF a_ant y position is less than 0
    Set a_ant y position to board height -1
  IF a_ant x position is greater than board width - 1
    Set a_ant x position to 0
  IF a_ant y position is greater than board height - 1
    Set a_ant y position to 0

Subprogram output()
  FOR each row on the Board
    FOR each column on the Board
      IF Ant’s position
        Output ‘@’
      ELSE IF Color is Off
        Output ‘ ‘
      ELSE IF Color is On
        Output “#’
End Class
```

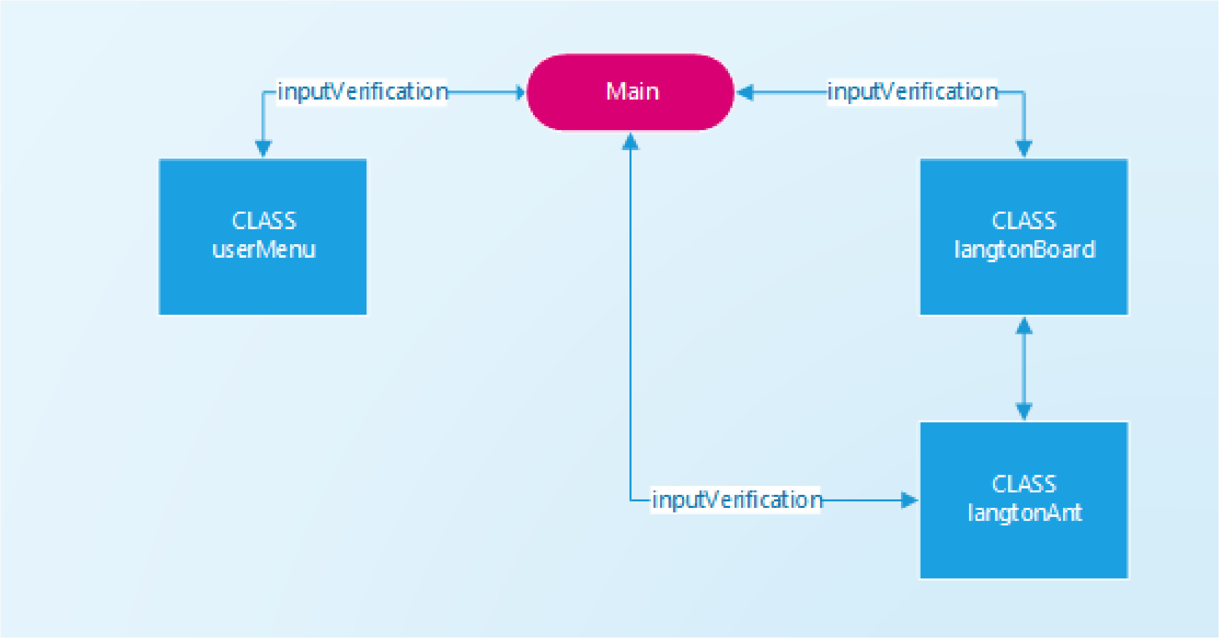
Class Designs

Class Name	userMenu
Data	choice (int)
Functions	makeSelection – User input for menu options

Class Name	langtonBoard
Data	WIDTH(int) – Board width HEIGHT(int) – Board height **a_board(int) – Board array ON(int) – Color On (constant = 1) OFF(int) – Color Off (constant = 0) *a_ant(Ant) – Ant object
Functions	void setAnt(Ant*) – Initialize Ant object for the Board void flipColor(Ant*) – Flip the color for the Ant object current position int getColorAt(Ant*) – Get the color for the Ant object current position int getWidth() – Gets Board width int getHeight() – Gets Board height void setWidth(int) – Sets Board width void setHeight(int) – Sets Board height void setBoard(int, int) – Removes/Initializes the dynamic Board array void moveAnt() – Moves Ant based on current position and color of cell void checkBounds() – Check the boundary and wrap Ant if needed void ouput() – Prints the Board

Class Name	langtonAnt
Data	Direction (enum) – Ant's enumerated direction values a_x(int) – Ant's x position on the board a_y(int) – Ant's y position on the board a_z(int) – Ant's number of moves a_direction(Direction) – Enumerated variable
Functions	int getX() – Gets Ant's x position int getY() – Gets Ant's y position int getDir() – Gets Ant's current direction int getMoves() – Gets number of moves Ant should make void setX() – Set Ant's x position void setY() – Set Ant's y position void setDir(int) – Set Ant's direction void setMoves(int) – Set number of moves Ant should make void turnL() – Turns Ant left dependent on Ant's current direction void turnR() – Turns Ant right dependent on Ant's current direction

Class Interactions



Test Case	Input Values	Driver Functions	Expected Outcome	Observed Outcome
Input too low	**Input < 0**	**inputVerification -> SafeInput()**	**Prompt user to enter correct value**	**Prompt user to enter correct value**
Input not an integer	**Input = 0d**	**inputVerification -> SafeInput()**	**Prompt user to enter correct value**	**Prompt user to enter correct value**
Input below range	**Input = -1**	**inputVerification -> SafeInput()**	**Prompt user to enter correct value**	**Prompt user to enter correct value**
Input above range	**Input = 100000**	**inputVerification -> SafeInput()**	**Prompt user to enter correct value**	**Prompt user to enter correct value**

Start simulation with default values	**menuOpt = 1
	ant-a_direction = 0
	ant->setX(20)
	ant->setY(40)
	ant->setMoves(80)
	board->setHeight(40)
	board->setWidth(80)**
		**board->output()
		board->moveAnt()**
			Simulation runs for 80 moves with a [40][80] board array and ant starting at [20][40] in the North facing direction	**Fail: The ant starting position was [40][20].**

Update board height/width and run simulation	**menuOpt = 2
	board->setHeight(15)
	board->setWidth(15)
	ant->setX(7)
	ant->setY(7)**
		**board->output()
		board->moveAnt()**
			Simulation runs for 80 moves with a [15][15] board array and ant starting at [7][7]	**Simulation runs for 80 moves with a [15][15] board array and ant starting at [7][7]**

Update ant starting position and run simulation	**menuOpt = 3
	ant->setX(0)
	ant->setY(0)**
		**board->output()
		board->moveAnt()**
			Simulation runs for 80 moves with a [40][80] board array and ant starting at [0][0]	**Fail: The ant starting position was out of bounds of the board array [-1][-1]**

Update ant starting direction and run simulation	**menuOpt = 5**
	ant-> setDir(2) – South
		board->output()
		board->moveAnt()
			Simulation runs for 80 moves with a [40][80] board and the ant starting at [20][40] in the South facing direction	**Simulation runs for 80 moves with a [40][80] board and the ant starting at [20][40] in the South facing direction**

Update ant number of moves and run simulation	**menuOpt = 6**
	ant->setMoves(25)
		board->output()
		board->moveAnt()
			Simulation runs for 25 moves with a [40][80] board array and ant starting at [20][40] in a North facing direction	**Simulation runs for 25 moves with a [40][80] board array and ant starting at [20][40] in a North facing direction**

Re-initialize ant and board objects and rerun simulation	**menuOpt = 7**
	ant-a_direction = 0
	ant->setX(40)
	ant->setY(20)
	ant->setMoves(80)
	board->setHeight(40)
	board->setWidth(80)
		board->output()
		board->moveAnt()
			Simulation runs for 80 moves with a [40][80] board array and ant starting at [20][40] in the North facing direction	**Simulation runs for 80 moves with a [40][80] board array and ant starting at [20][40] in the North facing direction**

Update board height/width and ant starting position off the updated board layout	**menuOpt = 2**
	board->setHeight(15)
	board->setWidth(15)
	menuOpt = 3
	ant->setX(25)
	ant->setY(25)
		board->output()
		board->moveAnt()
			Simulation fails to run and prompts the user for a valid ant starting position.	**Fail: Simulation fails to run and program crashes b/c ant position is out of the board array.**

Reflection

The week 2 project implementation was very challenging and I found multiple issues along the way. After initially implementing the Board and Ant in the same Class, it became obvious that they could be separated based on the separate functions they needed to perform. This allowed me to create a board object that was the primary object in the simulation. Everything after that was based on the manipulation of the board array.

I also found the array indexes and the user input caused me various issues in the beginning. I tried to initialize the array with values from my getHeight/getWidth functions. However, because I subtracted 1 from the initial Ant->setX/setY, I ended up negatively indexing the array. This also caused problems when I tried to move the Ant to the opposite side of the board for the wrap function. Once I determined the best way to dynamically populate the board array, the rest of the functions started falling into place.

I also made a large error in the beginning with my Public vs. Private object variables. I was setting a number of the variables to Public because I was trying to access them using the -> operator. This was to allow main to access a variable directly and I soon realized my error. The problem was I was accessing the public variable outside the class and needed to modify the program to account for the variables.

The other area where I struggled was with the initial position of the Ant. My default was to just center it in the Board whenever the array was allocated. However, when I allowed the user to define a different starting position, I was restricting the input based on the default board size. After I did some more work on the design, I changed my inputVerification to read in the board height/width and restricting the user to the current board values. I also took some liberties in assigning the Ant to the center of the board if the user opted to change the Board dimensions again.

The final (and most time consuming) issue I came across was clearing the dynamically allocated array memory. I was trying to do everything through my Main function and it appeared to work until I ran it through Valgrind. My initial design properly cleared the memory for the default board, but if the user changed the board dimensions, it failed to clear properly. I was having a hard time with the default constructor/destructor and how to dynamically clear the memory. Ultimately it came down to removing the referenced array memory after I was done using it. When the user selected to change the default board, I needed to clear the default array at that point. Once I created a new board object array, the destructor worked perfectly.

This project caused me a lot of headaches because I spent a lot of time refreshing my memory on core C++ principles. It was very rewarding to get a final product working and using the Object-Oriented design, I could implement new menu functions fairly smoothly. Tracing values through the code was where I refreshed most of my memory. While it was frustrating at times, it helped me get back into the use of pointers and arrays.