Brian Palmer
palmebri@oregonstate.edu
October 5, 2016
Design Document

Assignment 1 - Langston's Ant

Introduction

This game will begin by letting the user decide the width and height of their 2D array. Once that is determined, the user will be prompted to enter the number of steps the ant should take. Finally, the user will have the option to enter a starting location for the ant, or to let the program choose a random location for them.

During each step, the *Interaction* phase will update the board in the background. Once completed, the updated board will be brought forward to the user during the *Display Phase*. The step count will increase by one, and the program will start at the interaction phase again. These phases receive more attention below.

Interaction Phase

The interaction phase precedes the display phase so that the display phase shows the most up-to-date 2D array. Within this phase, an ordered sequence of events must occur. First, the direction the ant faces should be updated. This shall be calculated using the current direction the ant is facing and the internal color of the cell the ant is currently on. Once facing the correct direction, the ant will move to the next cell by updating the 'new' cell and 'old' cell accordingly. The internal color and Color members will be updated at this time.

After reading about Cellular Automation theory, many of these steps will be done utilizing the ants Van Neurmann neighborhood. The neighborhood is calculated by including cells that are are at least one step away from the ant in each orthogonal direction (https://en.wikipedia.org/wiki/Cellular_automaton). This neighborhood will be calculated at the beginning of the phase and erased at the end, and will be stored as a member in a 5 element Cell-Pointer array. Creating this subset makes sense because the ant can only move one space in any orthogonal direction. In addition, mutating member functions will be able to access the new cell by looking here versus the entire 2D array. While some kind of ordering process will need to be implemented on the array (each position will need to be associated with an orthogonal direction), more time would need to be spent to find these cells by searching the 2D array. Since the size of the 2D array is decided by the user, searching the array could impact run times.

Display Phase

During this phase, the updated 2D array will be shown to the user. The user will also see how many steps the ant has taken.

Plane Class

Private Members:

- Cell ** Field - Points to an array of Cell-pointer arrays.
- Van Neurman Neighborhood - This will be an array of pointer to cells that are within the van neurman neighborhood of the Ant. This array will be calculated during the interaction phase, and will be used to determine which direction the ant should face. The array will also be used to move the ant to a new cell, and will lastly be used to update the main 2D array. Finally, before the Display Phase, all the pointers will be nulled, ready to be used to create the neighborhood during the next interactive phase.
- Rows and Columns - The width and height of the 2D array (see field member)
- Cell * Ant - Cell pointer that will be updated each interactive phase with the Ant's address. This will be updated the same time that the van neurman neighborhood is updated. The member is there to easily access the ant cell.

Public Members:

- Setters and Getters for all members
- Initialize Ant Position() - this function will add the ant to the 2D array at the user's specified location
- Find Ant () - this function will be a utility to quickly find the ant in the 2D array
- Create Neighborhood () - This function starts by finding the ant, and then setting the pointers in an array to the cells within the ants van neurman neighborhood. The array is then set to the van neurman member.
- UpdateFacing() - This function updates the direction the ant is facing based on the internal color of the cell. This needs to occur before the ant can move.
- UpdateLastCell() - After the ant Moves, the last cells color and internal color will need to be updated
- Move () - using the updated direction the ant is facing and the internal color of the cell, one of the cells in the Van Neurman Neighborhood will be updated. Specifically, the color will change to '*'.

Cell Class

Private members:

- New Position // the position the ant will take during Update Phase. Ant will set during Interaction Phase. Precursor to setting this member is knowing which direction the ant will move, which has it's own precursor (what color the ant is standing on, which is stored in Current Color member).
- Internal Color // This will hold the current color of the cell. This member will be used to determine which direction the ant cell will face when updated.
- Color // will hold a ' ', '#' , or '*'character. This will be used to identify the cell with the ant on it, and will used when displaying the different cells on the 2D array
- DirectionFacing // this will be used to determine orthological direction of left and right // will either be N, S, E, or W

Public members:

- Accessors and Mutators

Menu Class

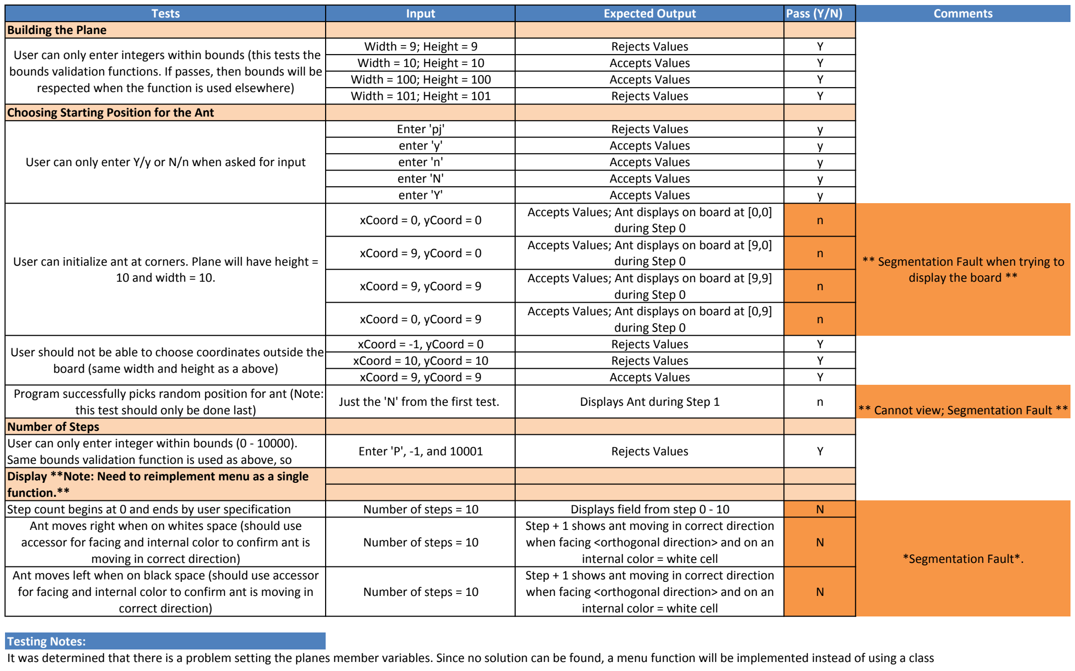
Private members:

- Plane-Pointer field // this member will point to the created plane class
- Num generations // will hold the number of steps that the user inputs
- xCoord/yCoord // will hold coordinates where ant should be initialized
- Width/Height // the width and height of the plane. These will be needed when determining where the ant can be initialized

Public members:

- buildPlane() // this function will get the width and height of the plane from the user, build the plane using the plane constructor, and then set the plane to the field member
- AskUserForNumGenerations() // this function will let the user choose the number of steps that they want the ant to take, and will set the number to the numGenerations member.
- RandomAntCoordinates() // this function will generate random coordinates that are within the planes height and width bounds, and then set the xCoord and yCoord members
- userDeterminesAntCoordinates() // this function will let the user enter their own coordinates within the height and width bounds, and will set the xCoord and yCoord members
- FacilitateStartingPosition() // This function will facilitate how the ant coordinates are determined. First the user will choose whether they want random coordinates (see RandomAntCoordinates()) or if they want to specify their coordinates (see userDeterminesAntCoordinates()). They will determine this with a Y / N response. Lastly, the function will initialize the ant at the determined coordinates by passing the values to the Planes initializeAntPosition() function.
- FacilitateIntInputWithinRange(int,int) // this function will include several validation functions to make sure the user’s input is correct.

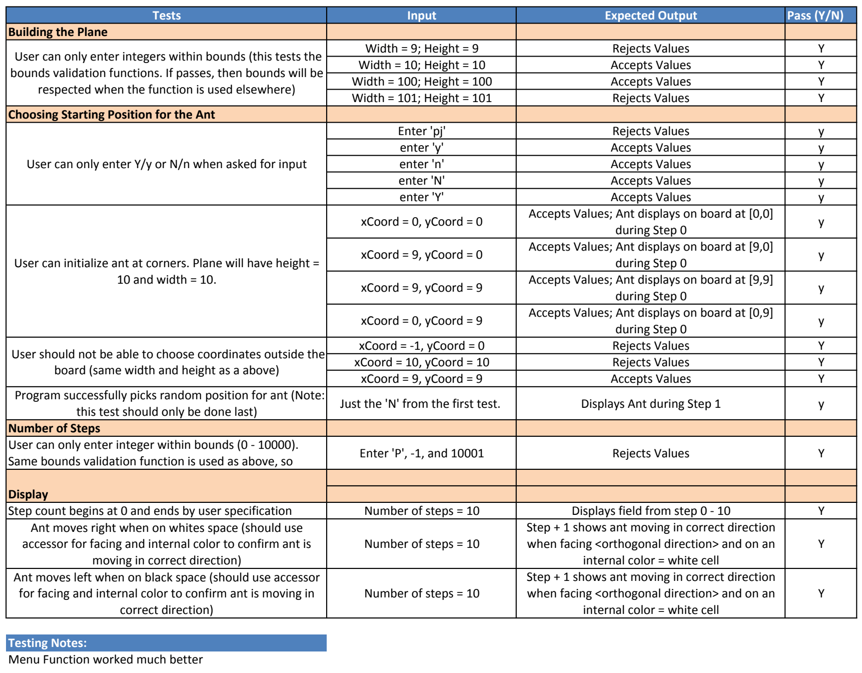
Tests:



Tests	Input	Expected Output	Pass (Y/N)	Comments
Building the Plane				
User can only enter integers within bounds (this tests the bounds validation functions. If passes, then bounds will be respected when the function is used elsewhere)	Width = 9; Height = 9	Rejects Values	Y	
	Width = 10; Height = 10	Accepts Values	Y	
	Width = 100; Height = 100	Accepts Values	Y	
	Width = 101; Height = 101	Rejects Values	Y	
Choosing Starting Position for the Ant				
User can only enter Y/y or N/n when asked for input	Enter 'pj'	Rejects Values	y	
	enter 'y'	Accepts Values	y	
	enter 'n'	Accepts Values	y	
	enter 'N'	Accepts Values	y	
	enter 'Y'	Accepts Values	y	
User can initialize ant at corners. Plane will have height = 10 and width = 10.	xCoord = 0, yCoord = 0	Accepts Values; Ant displays on board at [0,0] during Step 0	n	** Segmentation Fault when trying to display the board **
	xCoord = 9, yCoord = 0	Accepts Values; Ant displays on board at [9,0] during Step 0	n	
	xCoord = 9, yCoord = 9	Accepts Values; Ant displays on board at [9,9] during Step 0	n	
	xCoord = 0, yCoord = 9	Accepts Values; Ant displays on board at [0,9] during Step 0	n	
User should not be able to choose coordinates outside the board (same width and height as a above)	xCoord = -1, yCoord = 0	Rejects Values	Y	
	xCoord = 10, yCoord = 10	Rejects Values	Y	
	xCoord = 9, yCoord = 9	Accepts Values	Y	
Program successfully picks random position for ant (Note: this test should only be done last)	Just the 'N' from the first test.	Displays Ant during Step 1	n	** Cannot view; Segmentation Fault **
Number of Steps				
User can only enter integer within bounds (0 - 10000). Same bounds validation function is used as above, so	Enter 'P', -1, and 10001	Rejects Values	Y	
Display **Note: Need to reimplement menu as a single function.**				
Step count begins at 0 and ends by user specification	Number of steps = 10	Displays field from step 0 - 10	N	*Segmentation Fault*.
Ant moves right when on whites space (should use accessor for facing and internal color to confirm ant is moving in correct direction)	Number of steps = 10	Step + 1 shows ant moving in correct direction when facing <orthogonal direction> and on an internal color = white cell	N	
Ant moves left when on black space (should use accessor for facing and internal color to confirm ant is moving in correct direction)	Number of steps = 10	Step + 1 shows ant moving in correct direction when facing <orthogonal direction> and on an internal color = white cell	N	

Testing Notes:
It was determined that there is a problem setting the planes member variables. Since no solution can be found, a menu function will be implemented instead of using a class

Figure 1. My first test using a Menu Class. There was a segmentation fault that I could not resolve. I made a Menu function instead.



Tests	Input	Expected Output	Pass (Y/N)
Building the Plane			
User can only enter integers within bounds (this tests the bounds validation functions. If passes, then bounds will be respected when the function is used elsewhere)	Width = 9; Height = 9	Rejects Values	Y
	Width = 10; Height = 10	Accepts Values	Y
	Width = 100; Height = 100	Accepts Values	Y
	Width = 101; Height = 101	Rejects Values	Y
Choosing Starting Position for the Ant			
User can only enter Y/y or N/n when asked for input	Enter 'pj'	Rejects Values	y
	enter 'y'	Accepts Values	y
	enter 'n'	Accepts Values	y
	enter 'N'	Accepts Values	y
	enter 'Y'	Accepts Values	y
User can initialize ant at corners. Plane will have height = 10 and width = 10.	xCoord = 0, yCoord = 0	Accepts Values; Ant displays on board at [0,0] during Step 0	y
	xCoord = 9, yCoord = 0	Accepts Values; Ant displays on board at [9,0] during Step 0	y
	xCoord = 9, yCoord = 9	Accepts Values; Ant displays on board at [9,9] during Step 0	y
	xCoord = 0, yCoord = 9	Accepts Values; Ant displays on board at [0,9] during Step 0	y
User should not be able to choose coordinates outside the board (same width and height as a above)	xCoord = -1, yCoord = 0	Rejects Values	Y
	xCoord = 10, yCoord = 10	Rejects Values	Y
	xCoord = 9, yCoord = 9	Accepts Values	Y
Program successfully picks random position for ant (Note: this test should only be done last)	Just the 'N' from the first test.	Displays Ant during Step 1	y
Number of Steps			
User can only enter integer within bounds (0 - 10000). Same bounds validation function is used as above, so	Enter 'P', -1, and 10001	Rejects Values	Y
Display			
Step count begins at 0 and ends by user specification	Number of steps = 10	Displays field from step 0 - 10	Y
Ant moves right when on whites space (should use accessor for facing and internal color to confirm ant is moving in correct direction)	Number of steps = 10	Step + 1 shows ant moving in correct direction when facing <orthogonal direction> and on an internal color = white cell	Y
Ant moves left when on black space (should use accessor for facing and internal color to confirm ant is moving in correct direction)	Number of steps = 10	Step + 1 shows ant moving in correct direction when facing <orthogonal direction> and on an internal color = white cell	Y

Testing Notes:
Menu Function worked much better

Figure 2. Using a menu function instead, I was able to get rid of the segmentation fault. My thoughts on the segmentation fault can be read in the reflection section.

Reflection:

Creating the design document prior to coding was a huge help during this project. Specifically, I realized early in the project that I needed to read more about cellular automata. This is how I learned about Von Neurmann Neighborhoods, which helped me discover what other functions I would need and how to implement them. In addition, the design document forced me to walk through the problem incrementally. I initially saw the ants movement as a one-step process, but realized that the problem was much easier to tackle in two-steps (update the direction the ant is facing, then move). In the end, the time I spent designing my Plane and Cell classes paid off because I was able to meet all of the requirements and only needed to debug syntax errors.

In my design, I also wanted to make a Menu class but ran into a segmentation fault when the program tried to display the field. After iteratively going through the steps, I was lead to believe that my user entered width and height values, and the constructed plane object in the BuildPlane() function were not being passed to the Menu members correctly. Unable to resolve why these members were not being updated, I decided to reimplement the menu as a single function. I was able to recycle many of my menu functions with some minor tweaking, so this did not cost me much time. My program worked much better with this new menu.