

DESIGN & REFLECTION DOCUMENT

SANDRO AGUILAR
LAB 3 SECTION 400

February 02, 2019

DESIGN

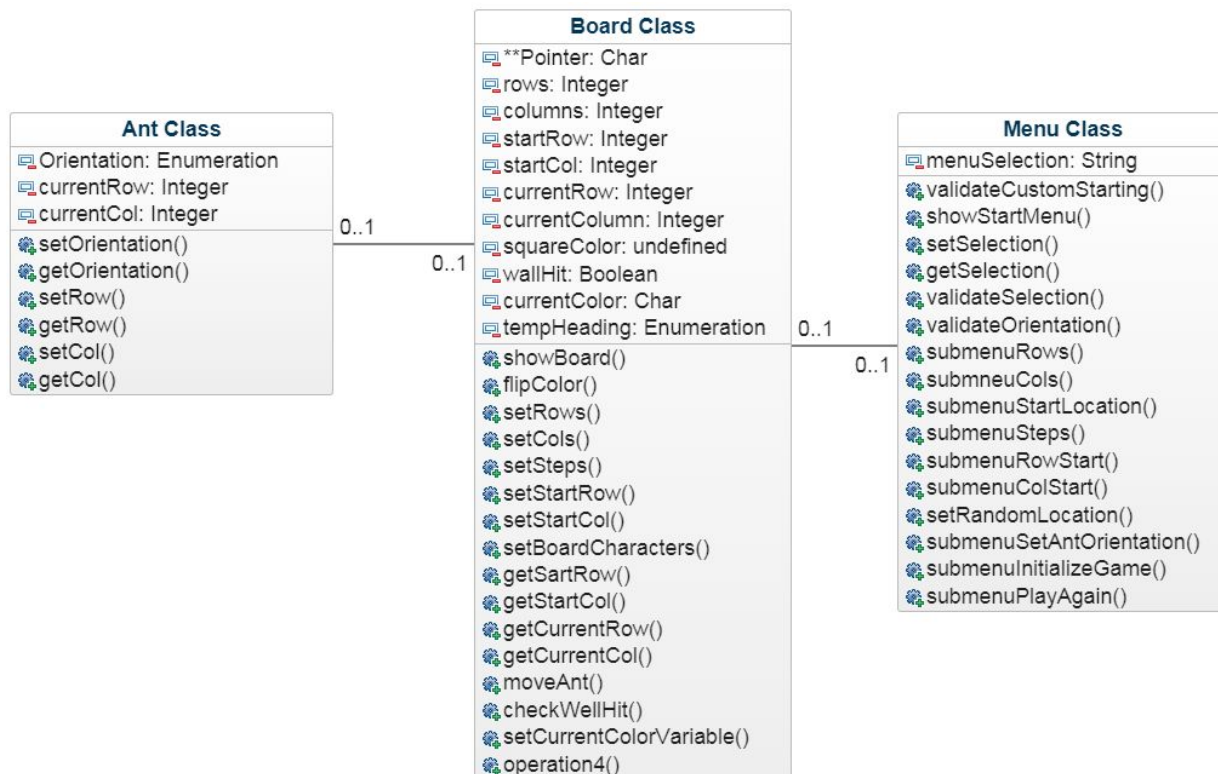
The overall design that I plan based on the project requirements will consist of three classes. We are required to have an Ant class however I have decided that in order to increase the modularity of my program and to create modules that I can later reuse, I decided to have a Board class and a Menu class.

I chose to have a board class because I felt like the board has many functions that only relate to the board itself which would clutter up the Ant class or Menu class. Also, having all of the Board code in main makes following the game logic difficult. I think it is good to have a clean main function where only calls to specific functions are made. For example, in my main, I have a function called showBoard() that shows the board whenever I call it; it doesn't matter how it does it, all someone cares about is that it shows the board when you read that line of code in main. This line of thinking applied to a lot of the code that I placed inside of the classes. However, I feel like there is code in my main that I could move into my classes however, time constraints really limit my ability to refine my design.

I also added a Menu class because I did not want to have all of this code in main either. Also, by having my menu in a class, I will be able to modularize my code so that I could later reuse it if needed. My menu basically prints out menu questions as needed during the execution of the game. It also holds validation functions for user input since it makes sense to validate user input when asked by the prompts in the Menu class.

Lastly, the Ant class was a required class of the project and it stores information such as the ant's orientation, its current row and column location, and setters and getters for the data members.

Below, you can see a rough draft of the classes that I envision and how they relate. The classes are associated with one another however there is no inheritance hierarchy or object composition.



The main function will hold local variables and possibly constant variables to help with validation. The main should create an instance of my menu class and it will call the menus during the appropriate times of the game. For example, at the beginning of the game, it should show the main menu.

Since the user will be required to make a series of choices to initialize the game, I image that there will need to be control statements in order to control the flow of the game such as moving from menu to menu as user enters data as well as exiting the game when given the option to do so.

Also, the game asks that the ant move a certain number of times, depending on the user's input. This necessitates the use of some sort of loop to control the movement of the ant.

A few design challenges that I can already foresee is determining how to set up the logic for checking if the ant is moving out of bounds. I feel like the approach I am going to take is to make the ant turn around 180 degrees if it moves out of bounds i.e., the ant bounces off the wall and now faces the opposite direction. However, this situation can get tricky in cases where the board is only 1 row or 1 column across. I will need to add logic to check for this as well.

Another challenge that I am currently wondering about is how to show a * character that represents the ant while keeping track of the color of the square it currently resides in. This may call for the use of a variable to save the color of the square when the ant lands on a square. However, I have to remember to recall the color of the square in order to change it the appropriate color after the ant leaves the square.

As the ant moves, my main function will be in charge of calling a function that will show the board with the current location of the ant and the current colors of the squares on the board. I plan on creating a 2D array using pointers that will store the char data type in order to store ' ' blank characters, # black characters, and * the ant character. Working with the char data type should make this easy.

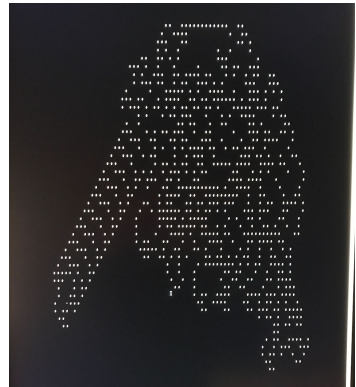
Since my 2D array is representing the board, it follows that it should be dynamically created in the board class so I will include a char pointer to a pointer as a private data variable and then dynamically allocate it with a function within the board class. It also follows that I should deallocate the array using the Board class destructor while carefully testing to make sure there are no memory leaks.

TEST TABLE

This test plan is provided to show how I tested the program. My goal was to create a test plan that is robust enough to provide assurance that the program behaves like it is meant to. The bulk of the test plan makes sure that the game parameters are properly set as this is what drives the rest of the program.

Test Case	Input Values	Expected Output	Actual Output
Main menu	Input == 1 Input == 2 Input < 1 Input > 2 Input = any characters, spaces, and number combination	Selecting 1 continues the game Selecting 2 ends the game	Game continues when 1 is selected Game ends when 2 is selected
Enter board Rows and Columns (1 min, 100 max)	Input < 1 Input > 100 Input == char char == any combination of numbers, characters, and spaces	Entering a value within the indicated range should set the row and column width	Row and column width only take numbers 1 through 100 *note- the max size of 100 was chosen arbitrarily for my program. It can be changed by modifying the const values in main().
Enter total steps to move ant (1 min, 20000 max)	Input < 1 Input > 20000 Input == char char == any combination of numbers, characters, and spaces	Entering a value within the indicated range should set the number of steps the ant should move	Steps only accepts digits 1 - 20000 *note- the max size of 20000 was chosen arbitrarily for my program. It can be changed by modifying the const values in main().
Select ant location - 1 for user picked location and 2 for random	Input < 1 Input > 2 Input == char char == any combination of numbers,	1 - should take the user to the menu asking them to manually set the row and column 2 - should take	1 - takes user to the menu asking them to manually set the row and columns 2 - takes user to the set ant orientation menu

	characters, and spaces	user to the set orientation menu	
Select Starting Row and Column location for the ant	Input < row Input > row Input < col Input > col Input -- char char == any combination of numbers, characters, and spaces	Row - should only take a number from 1 up to size of Row chosen in step 2 above Col - should only take a number from 1 up to size of Col chosen in step 2 above	Row - only takes a value from 1 up to Row Col - only takes a value from 1 up to Col
Select Ant Orientation Enter 1, 2, 3 or 4	Input < 1 Input > 4 Input == char char == any combination of numbers, characters, and spaces	1 should set the orientation North 2 should set the orientation South 3 should set the orientation East 4 should set the orientation West	1 sets North 2 sets South 3 sets East 4 sets West All other input is rejected
Show Board	N/A	The board should display with the ant located in the selected location	The board is properly displayed
Ant moving out of bounds	Place the ant on a 3x3 board and have it start at row 3, col 3 (bottom corner) with an orientation heading of North	The ant is facing North so it should move right. Right is out of bounds so the ant should turn around 180 and move West one column. The space it was previously in should change to black and the ant should base its next move on its new heading and color of the square it moved to.	The ant tries to move right however it is out of bounds so instead it moves left now facing west. The spot it was previously at is now black. The moved to a blank spot and since it is heading west, it should turn right since it landed in a blank spot. Ant moves right.
1 x 5 board	Place the ant on a 1x5 board and have it face West so	The ant is facing West on a board consisting of 1 row and 5 columns.	The ant tries to move North however it cannot so it moves South, which it cannot so it stays in

	that it tries to travel North	Since it is facing West, it will try to move North. However, since there are no more rows, it should instead bounce off the north wall 180 and face south. The ant should not move from its current location. Since its current location is now black, it should turn left which in this case is East.	the same location. The current square color is not black so on the next move, the ant moves left.
1 x 1 board	Place the ant on a 1x1 board	The ant should not move out of bounds. No errors should occur.	No errors occur.
Board Pattern	100 x 100 board, ants starts on row 50, col 50, set the steps to 11,500	The ant should form the typical pattern if it is properly working like it should.	Pattern emerges as shown in picture shown: 

REFLECTION

As I worked on my project, I typically do not have all of the exact details of every single variable or member function that I will need. Sometimes, you just have to start coding and then you'll see what you need. However, having a model like the one created during the design process helps in creating your overall structure for your project. I have found that as long as your project

is structured well, then it is a lot easier to complete without any major rework.

I think the biggest challenge for me was thinking of a system of tracking the color of the square the ant was currently on. So what I did was create a variable that stores the color of the square the ant moves to and once the ant moves again, another function uses this variable to determine what color to flip it to. This part was difficult for me because there are a lot of things to do when the ant moves.

Another challenge I had was when it came to edge cases. How should the ant behave when the board is 1x1 or 1x10? My program was structured so that when the ant moves, it first looks at the ant heading and then at the board color it is currently on. This is what is required to move the ant left or right. It then sets the ant heading and will proceed to move to that location. However, in these edge cases, there may be no room to move to if it is heading out of bounds so I created a function called `moveAnt` which takes into consideration if the move would make the ant out of bounds. This behaviour applied is to instead make the ant turn around 180 degrees and then move by one. However, if there is not room to move in either direction, the ant does not move but instead just rotates since the color it is on is changing from black to white which causes the ant to change its heading left or right.

On a good note, I was glad I worked on getting my validation functions ironed out in the first few labs because I was able to reuse those functions for this project which saved me considerable time.

I definitely learned a lot with this project. The biggest areas I gained experience in was working with arrays and finding ways to make sure the ant moved according to the rules of the board without any erratic behavior. My favorite skill I learned for this project was working with the pointer to pointer of arrays and how to deallocate the memory while keeping it all in a class. It required a lot of hours of work just for that part however it was well worth it.