**ALGORITHM SIMULATOR**

Naser Anjum

1505099

Project Functionality:

The project, Algorithm Simulator, simulates various algorithms through animation. In this project, the algorithms that have been included are-

1. Bubble Sort
2. Selection Sort
3. Breadth First Search (BFS)
4. Depth First Search (DFS) and
5. Binary Search.

For simulation, Java Transition class has been used. The used Transition subclasses were-

* Translate Transition (to translate a node)
* Fill Transition (to change the color of a node)
* Scale Transition (to increase or decrease the size of a node)
* Fade Transition(to fade the color of a node)
* Parallel Transition(to run other transitions in parallel)

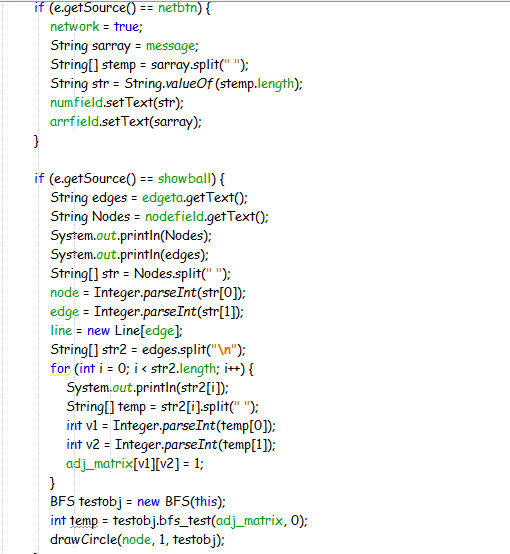
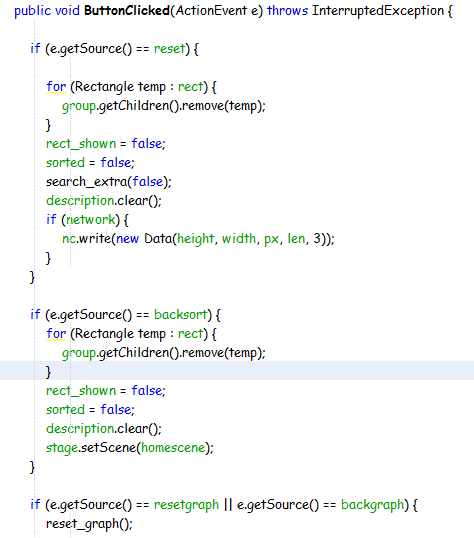
In the sorting simulations, the user needs to input the array size and the array elements. Rectangles are drawn and the height of the rectangles are set according to the elements. Then, if the sort button is clicked, comparing and swapping starts according to the algorithm. When two elements are compared, Fill Transition has been used to distinguish those rectangles from others. In case of swapping the rectangles, Translate Transition has been used. The description text area briefly describes the state of sorting. Some networking has also been added to the project. The client user can also use the main program to sort the array and simultaneously simulate with main program.

After the array has been sorted, the user can binary search any element. Initially in the binary search, the start, middle and end points have been distinguished by color change. Then, the part that we won’t search in, will be faded and the new start, middle and end points will be shown. If found, the rectangle will be shown in Red color.

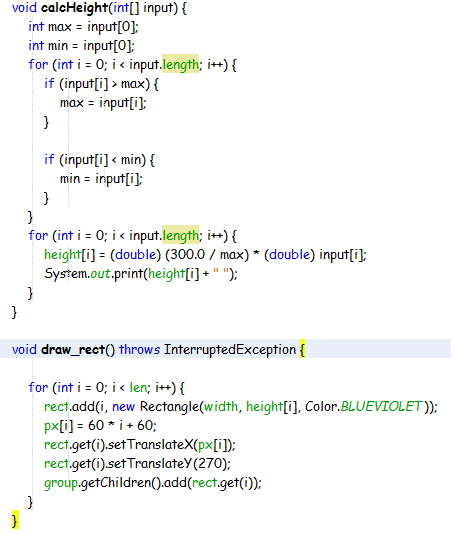
In Graph Algorithm simulation, the nodes and edges have been represented by Circle and Lines respectively. There is a pre-defined graph set and also the user can input the graph size and the edges. When the graph Traversal starts, the current node is highlighted by Scale Transition and the color changes according to the level that node is in. In traversal time, the edges are shown simultaneously by dotted lines.

CODE SNIPPETS

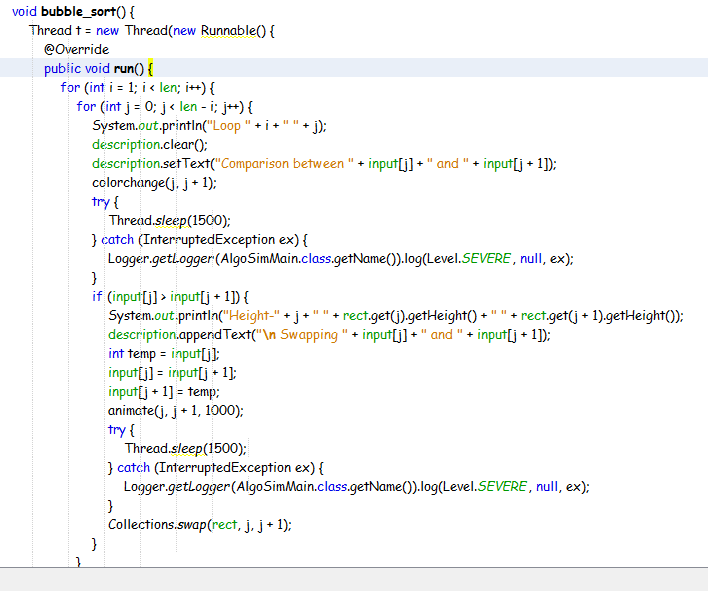
* The ButtonClicked() function is called whenever a button is clicked. In this function the needed action has been written for each button.



* Calc\_Height function calculates the height of the rectangles according to the array elements. The draw\_rect function adds rectangles to the group. Here, List has been used.



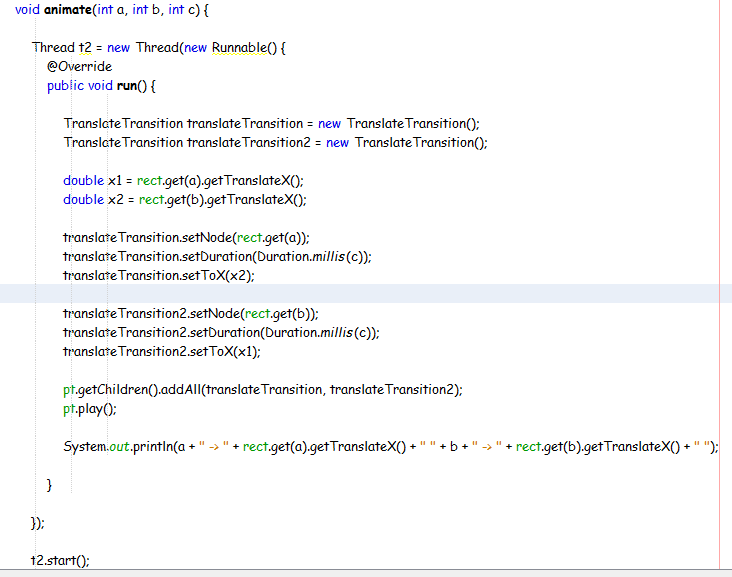
* The bubble sort function sorts the array. It runs on a thread that is separate from the animation threads, that’s why slowing down the sorting thread doesn’t affect the animation process. While comparing, it calls the colorchange() function and while changing the position of rectangles it calls the animate() function.



* The selection sort function works almost same as the bubble sort one, but the comparing process is different.



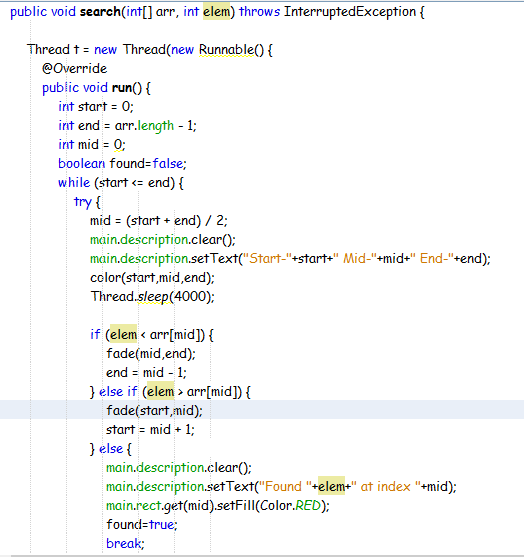
* The animate() function takes the index of the rectangle list that needs to be swapped and also the duration. It takes two Translate Transition objects and adds those to Parallel Transition for playing the animation at the same time.



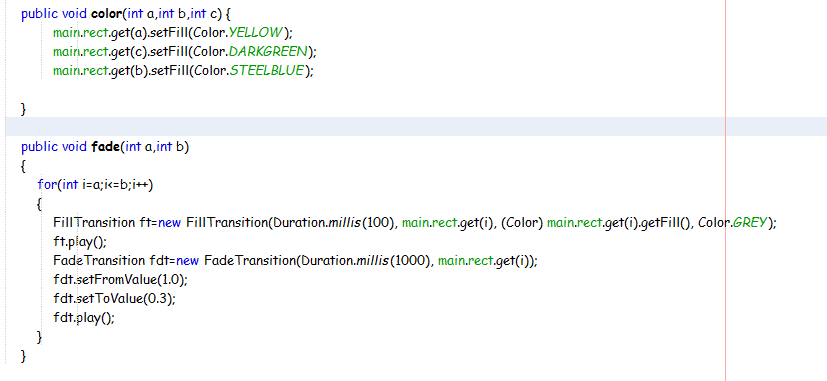
* The colorchange function changes color of the rectangle of given index through animation, but reverses back to the original color after comparing has been done.



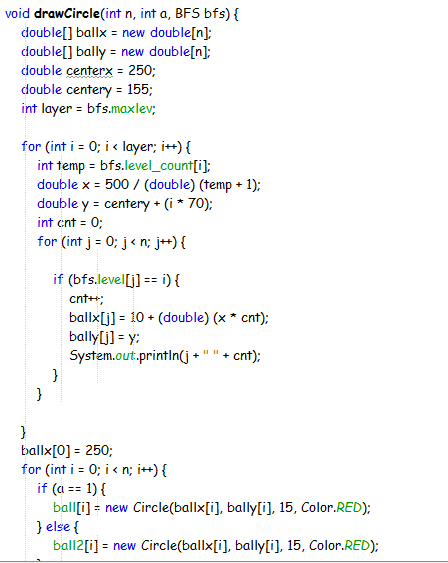
* The search function in BinSearch class takes the array and the element to be searched as parameters. It uses color and fade function for changing the color effectively.



* Fade function just fades the color and color function colorizes the rectangles at staring, middle and ending index.



* It draws the circles positioning them by levels which has been determined by a simple BFS traversal before the original simulated traversal.



* In BFS, queue is used and whenever a number is added to the queue, the Circle corresponding to the number gets highlighted and if it is connected with another node, a line is drawn to represent edge.



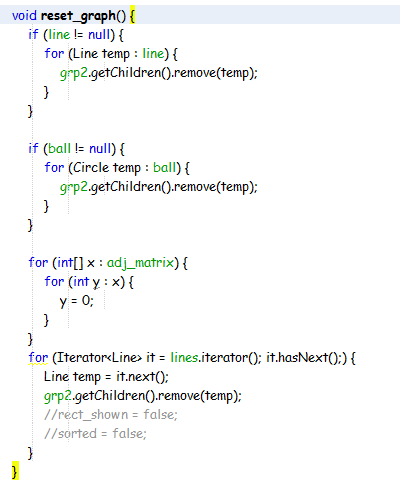
* In DFS, instead of queue, stack is used and the animation process is almost the same as BFS.



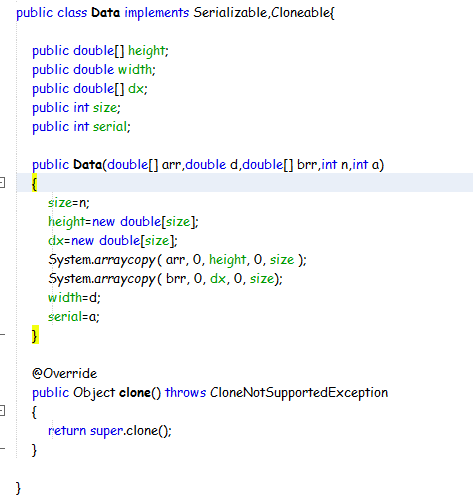
* In the animate() function graph, it changes color, increases the size of color and draws a line after color change.



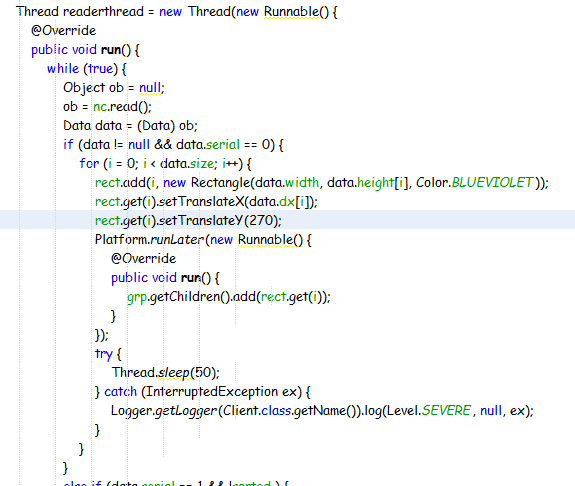
* The reset function removes some shape objects to make the scene return to initial stage.



* The Data class has the information needed for rectangles to be drawn and animated via networking.



* The readerthread reads the data and as per data information it performs action that is called in its run method.



Things I could have done

In my project description, I mentioned that I would like to do some sorting, graph algorithms and data structure simulations. I did the sorting and graph algorithm part but could not do any data structure visualization because of time constraints. I thought of doing Recursion Tree, Recursion Graph (DP) or the Binary Heap which I could have implemented using some animation that I used in case of BFS, DFS. Overall I felt good to implement the part of simulating the algorithms which at first I thought would not be possible.

Difficulties I have faced

The first problem I faced was at the time of making the GUI with scene builder. It worked in some cases but often it used to show exceptions and this prompted me to shift to plain coding in JavaFX style. It needed some time to code these but it worked without much problem.

Making the first simulation was the most difficult part and it took me some time to make Bubble Sorting work. Initially, the translate transition was working but Rectangle property swapping was not working properly and that’s why animation was not syncing properly. Then, instead of manually swapping each property of the rectangles I took a list of rectangles and used Collection.swap(). It solved the problem.

Then, in Binary Search implementation, the color change was not syncing properly because of improper use of thread and its sleep method. While implementing the networking part, the problem was the shape objects are not serializable and could not be sent as data through socket. So, the rectangle properties were sent and using those properties the rectangles were drawn and animated.

Because of difficulties faced in implementing the simulation part, I could not concentrate much on improving the graphical user interface.