Minwoo Soh

Professor Dominick Atanasio

CS 4200-01

3 March 2019

Project 1: 8-Puzzle

While working on the project, the first implementation was the Node class. The “puzzle” array was used as the state. I did not use a 2d array, so the puzzle array is set up with 9 entries and every 3 entries considered a row. Indexes 0, 1, 2 is the first row, indexes 3, 4, 5 is the second row, and indexes 6, 7, 8 is the third row. The entries are non-duplicate integers from 0 to 8 representing each tile numbers. If the puzzle was complete, then it would be 0, 1, 2, 3, 4, 5, 6, 7, 8, which shows that the indexes and the entries line are equal to each other if it is complete.

There are 2 separate nodes, “parent” and “next,” stored inside the node object itself. Parent is used to link the nodes to their previous actions, while next is used for the Priority Queue for frontier queue and for the explored node list. The variable “pathCost” is used for depth counter. The variable “fValue” is used to store the f(n) = g(n) + h(n) value.

Implementing the function for the h1 heuristic was simple. If a puzzle array entry did not match with the entry’s index, then it is misplaced. The program simply had to go through each entry in the puzzle array, and if the index did not match up with the entry integer, then it would increment the counter.

Implementing the function for the h2 heuristic was much harder since I was using a normal array. As result, I had to think of different ways to account for vertical and horizontal distances from original position. Since rows were divided by every 3 entries in the array, my implementation required the use of modulus. I realized that if I divided each column indexes by

3, the remainders in the column matched. For example, the first column, which consists of the indexes 0, 3, and 6, have remainders of 0. The second column, which consists of the indexes 1, 4, and 7, have remainders of 1. Therefore, I was able to get the horizontal distance by doing the operation: Math.abs(entry%3 - index%3). Getting the vertical distance required a similar approach. I realized that subtracting a tile number by the remainder of that number after dividing it by 3 (tile number – tile number%3), it would return the index of the first column of its respective row. By doing the operation, Math.abs((entry-entry%3)-(index-index%3))/3, I was able to obtain the vertical distance. Adding both horizontal and vertical values would result in the Manhattan distance from a tile's current location to its goal positions. This was done with in a for loop for each entry in the puzzle array, and the totals for each tile was added to get the h2 heuristic value.

Test done with 2000 test cases for each depth:

	A* average search costs/times			
Depth	h1 cost	h1 time (seconds)	h2 cost	h2 time (seconds)
2	6	1.97E-06	6	1.04E-06
4	10	4.60E-06	10	2.98E-06
6	17	6.95E-06	18	5.15E-06
8	31	1.86E-05	30	1.23E-05
10	70	6.20E-05	60	3.10E-05
12	201	2.57E-04	142	9.45E-05
14	517	0.00137	319	3.59E-04
16	1324	0.00979	742	0.0014
18	3221	0.0696	1587	0.0051