Ryan Le rle026
Aidan Lopez alope396
Brandon Tran btran117
Vy Vo vvo025

CS170: Project #1 8 Puzzle Report

Challenges:

Our main challenge was designing the structure and object classes. Initially, we created a tree class to build a tree and search through the tree; however, we figured that it would be easier to do this in the problem class and make the tree as we expand the nodes.

Another challenge was understanding what each algorithm is supposed to do and the difference between the different types of search. Some of us were confused by the uniform cost because the cost from one state is 1. We resolved this by considering the depth of the node, so we would add 1 to the cost when we go down the tree.

In addition to this, some of us get mixed up between Euclidean and Manhattan distance heuristics. We learned that the Euclidean distance heuristic is the shortest path from the current index to the target index, so we would calculate the Pythagorean (hypotenuse) for the distance between the current tile from the target tile.

We also utilized helper functions to move the tiles while our code ran. While the tiles did move, there was an error in how these functions were implemented, causing our heuristic values to be calculated incorrectly. After cleaning up some of the code and making sure the values were assigned properly, we were able to fix our heuristic calculations.

Our code also encountered an issue with getting stuck in infinite loops when choosing the best path. To solve this, we first came up with an idea of keeping track of the previous move, thus preventing the tiles from sliding back and forth forever. However, while debugging other issues, we came across the idea that the loops could be much bigger (ex. The tiles move in a circular path around a 2x2 space within the puzzle). To prevent this from happening, we removed the previous move tracker and instead implemented a set of states that were already tested. This way we could check each board state to see if we had already encountered it and then terminate that node before pushing it to the frontier if necessary.

Finally, our group wanted to test other cases outside of the ones provided to make sure our code was correct, however we did this before we included the check to see if it was impossible. This coupled with the sample tests having a different goal state (which we did not realize until much later), caused us to comb our code to figure out why it could not find a solution.

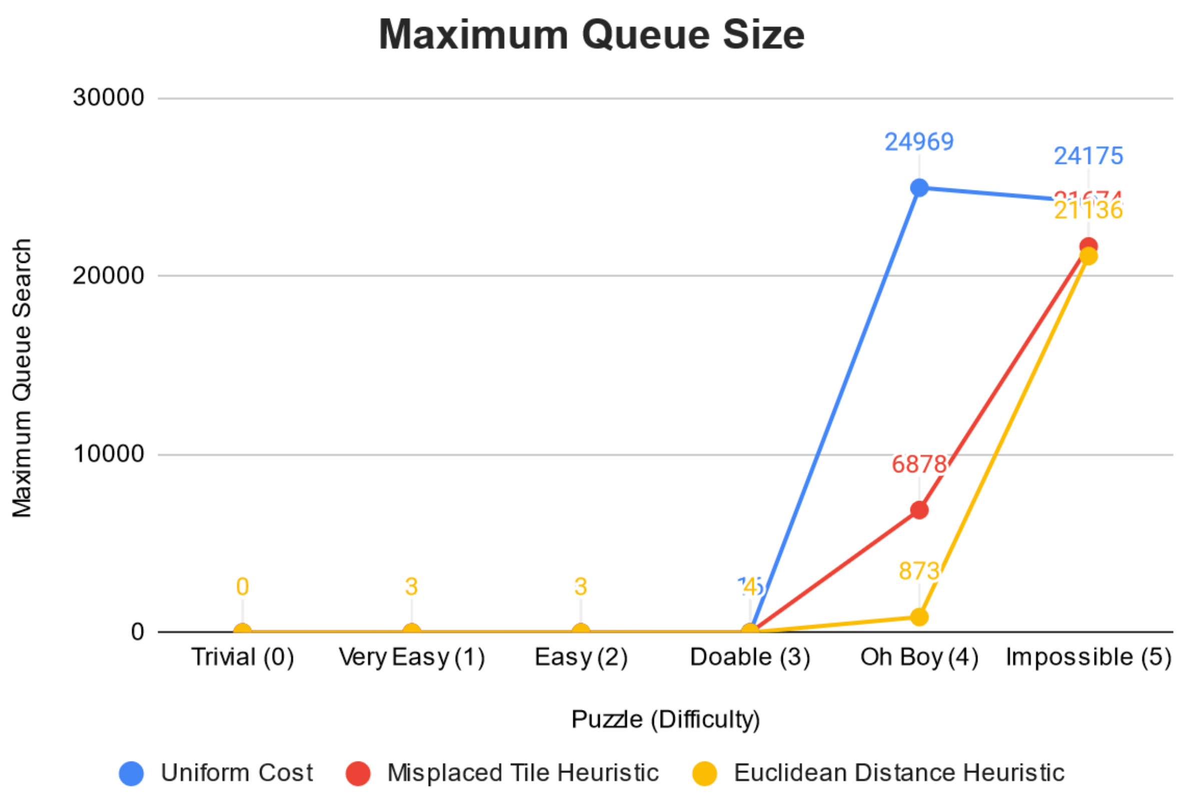
Design:

We created a problem class and a node class. The problem class has details about the problem which includes the initial state, the goal state, the operators, and the search algorithm. For state representation, we are using a 2D vector to set the puzzle at each state since the puzzle is a 2D matrix. For indexing, we are using pairs to get the position of each tile. We also created a Tile structure to store information for each tile, which includes the current index, target index, and value. This information is needed for the search algorithm to use the operator to change the current index of the tile and for calculating the number of moves it takes for the tile to move to its target index via the Euclidean method.

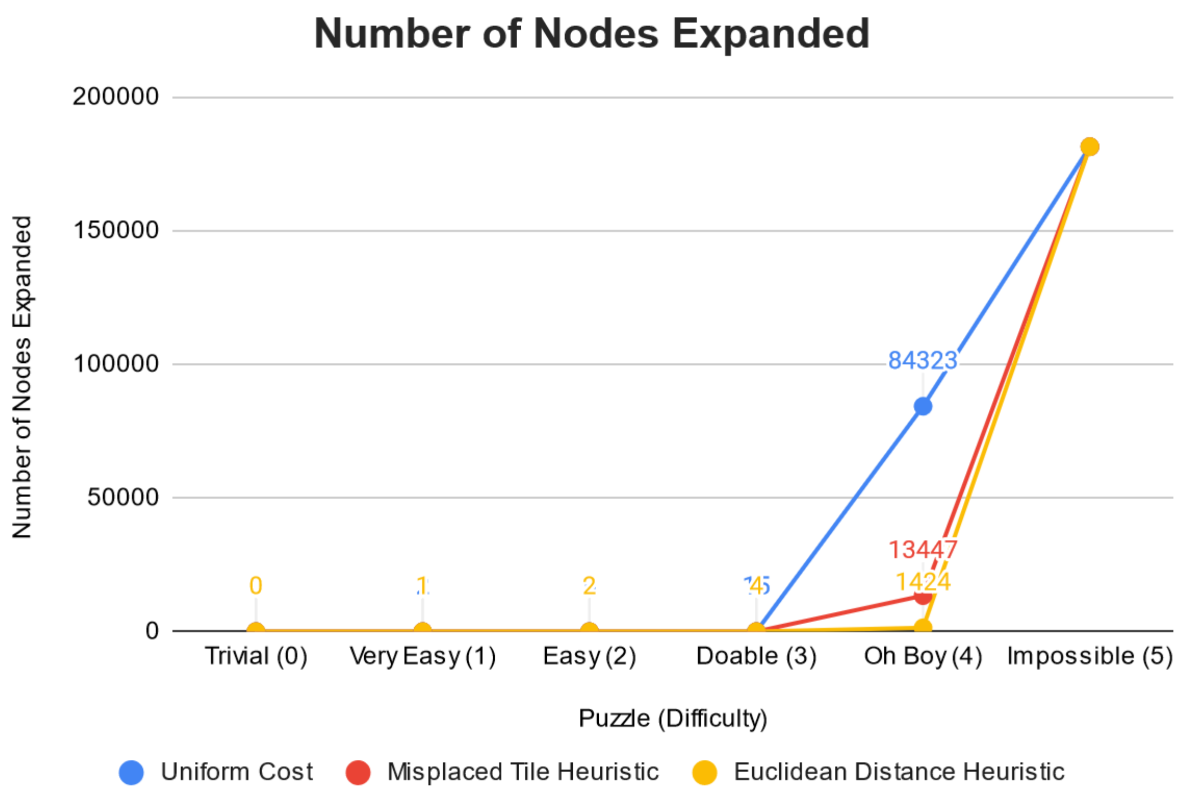
For the search algorithm, we are using a priority queue STL as the frontier to get the nodes and check if the goal state has been reached. Before adding a node to the frontier, we add them to a set to keep track of the states we have already tested. Therefore, if the node cannot be added to the set, we have encountered a loop and thus do not push it to the frontier.

Heuristic Functions / Findings:

Maximum Queue Size			
	Uniform Cost Search	Misplaced Tile Heuristic	Euclidean Distance Heuristic
Trivial (0)	0	0	0
Very Easy (1)	3	3	3
Easy (2)	6	3	3
Doable (3)	16	4	4
Oh Boy (4)	24969	6878	873
Impossible (5)	24175	21674	21136



Number of Nodes Expanded			
	Uniform Cost Search	Misplaced Tile Heuristic	Euclidean Distance Heuristic
Trivial (0)	0	0	0
Very Easy (1)	2	1	1
Easy (2)	5	2	2
Doable (3)	15	4	4
Oh Boy (4)	84323	13447	1424
Impossible (5)	181440	181440	181440



Upon inspection of the graphs, the most efficient method of finding the solution is using the Euclidean Distance Heuristic. For the "trivial" to "easy" puzzles, the graphs differ slightly, with UCS having a slightly higher number of expansions and larger queue size. However, this gap grows significantly as it moves from "doable" to "oh boy". The "oh boy" category shows UCS with the highest number of expansions and largest queue size, which is to be expected since it does not take heuristics into account. While knowing the number of tiles that are misplaced and taking that into consideration when moving is an improvement from UCS, the euclidean distance heuristic is far more superior, reducing the expansions of MTH by a factor of 10. From these two graphs, we can conclude that the addition of a heuristic is indeed impactful to the algorithm and is superior to the greedy UCS algorithm as the solution becomes harder to find.