Cris Liao, Aishani Patalay, Rayyan Zaid , Kartik Gulia

CS 170

Project 1

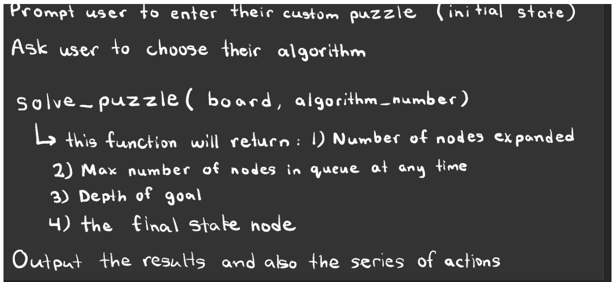
5/6/2023

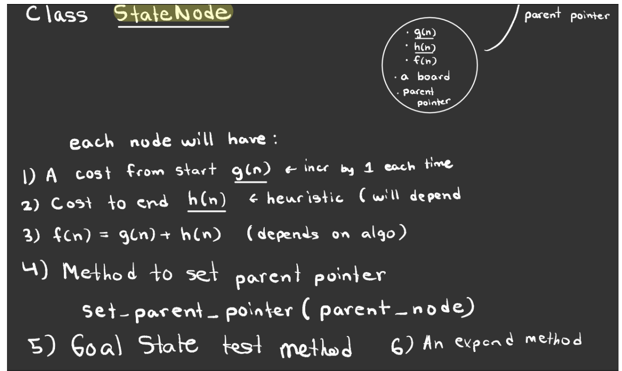
Github Link: 8-Puzzle Repo

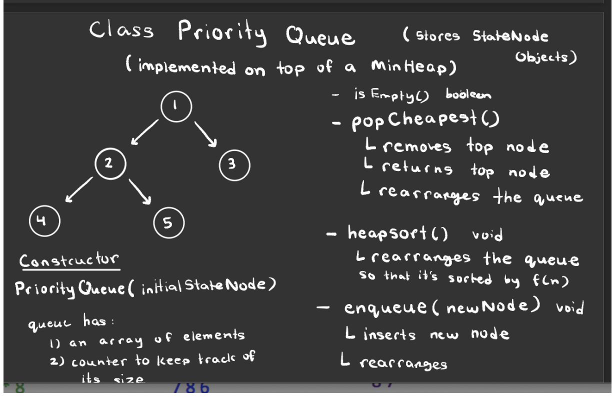
Project Report

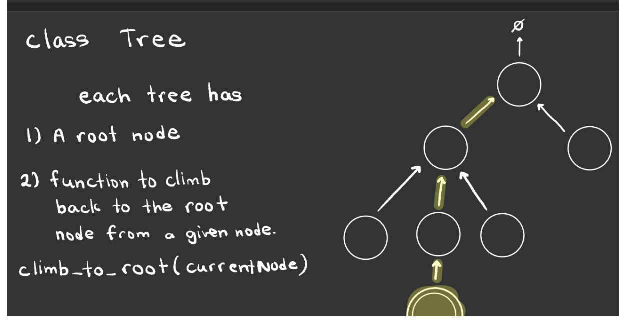
Some of the challenges that our group faced during this project was planning the design for everything. Initially, it was difficult to put everything together because of how many different components we had. However, after planning mostly everything went smoothly. Another challenge we faced was deciding all the properties of the StateNode class. Looking at it superficially, it may seem like each state is just a board, but after going through the coding process, we realized that there are so many characteristics of a node that allow it to function and give it meaning. Additionally, waiting for the impossible case to run took quite a bit of time. The program took twenty minutes to run before getting stuck. One more challenge we faced was deciding how to complete the extra credit. For this, we had to add more features to the statenode to ensure that there was a defined path back to the initial node.

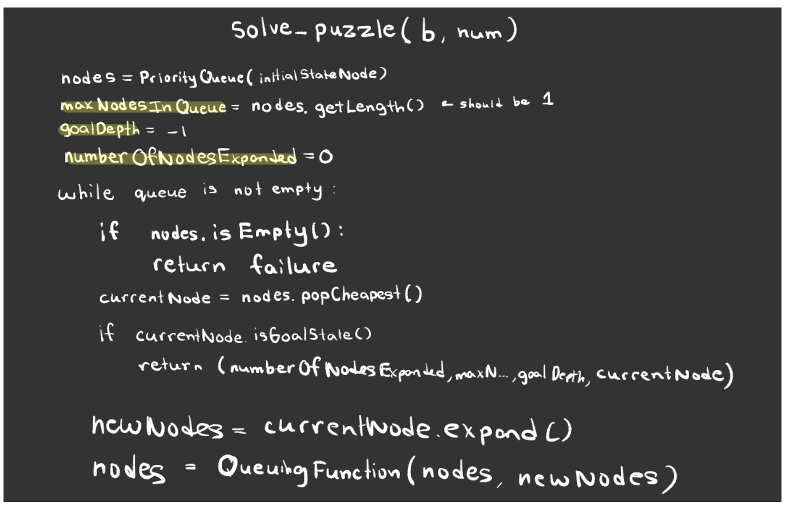
Design of the Algorithm: We designed our code to have the user enter the initial state of the eight piece puzzle and enter the type of algorithm they want to use to solve the puzzle. They can choose from one of the three algorithms (Uniform Cost Search, Misplaced Tile heuristic, and Euclidean Distance heuristic).





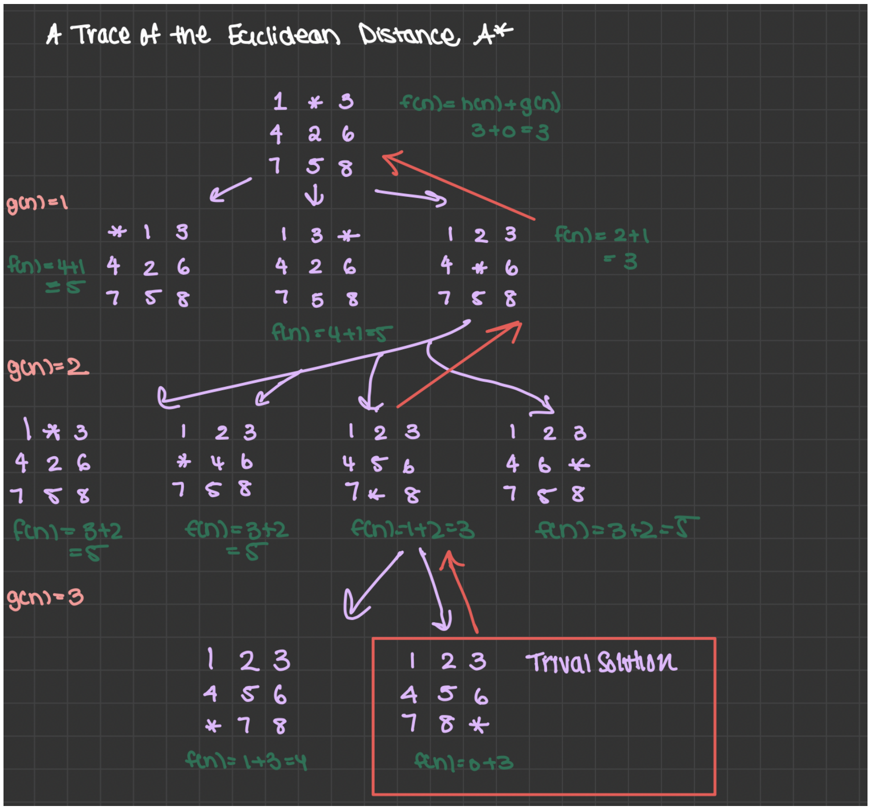






We try to optimize our code by using a priority queue (built-in) that makes searching for and comparing states faster. To make the search faster, we had to trade memory. What we did was create a set of nodes (visitedSet) and added each node to it when it was expanded. So then when we searched new nodes and they ended up being duplicates, the program could determine that in constant time. We have a statenode class file for creating nodes and tracking g(n), h(n), and f(n).g(n) is the function with cost of the start, and it increases by one each time. h(n) is the heuristic function that computes the cost to the end. f(n) is the addition of the g(n) and h(n). We used a tree search algorithm.

A Trace of the Euclidean Distance A*



Sample Test Cases:

Some test cases:

Project 1

Test cases

Trival

1 2 3
4 5 6
7 8 *

Easy

1 2 *
4 5 3
7 8 6

Oh Boy

8 7 1
6 * 2
5 4 3

Very Easy

1 2 3
4 5 6
7 * 8

doable

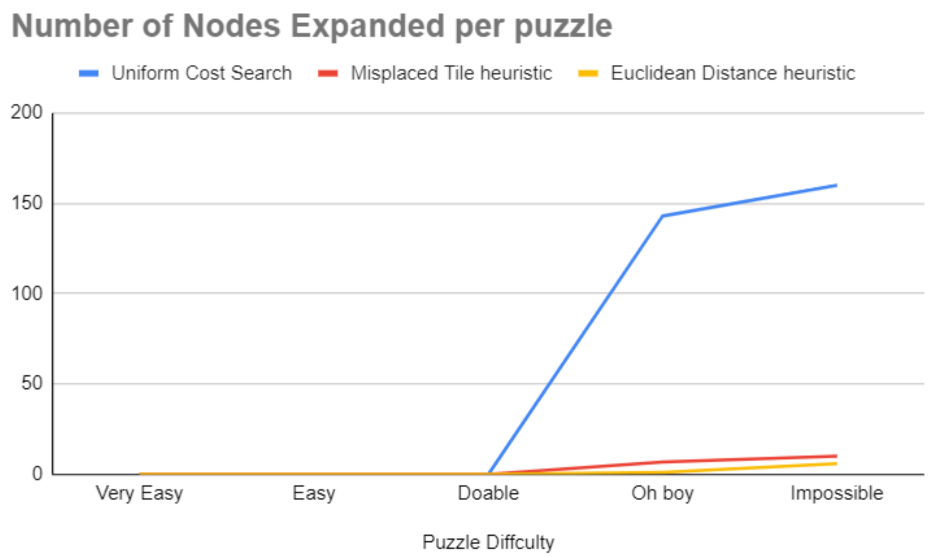
* 1 2
4 5 3
7 8 6

IMPOSSIBLE: The following puzzle is impossible to solve, if you *can* solve it, you have a bug in your code.

1 2 3
4 5 6
8 7 *

Sample diagrams and table:

Node expanded:



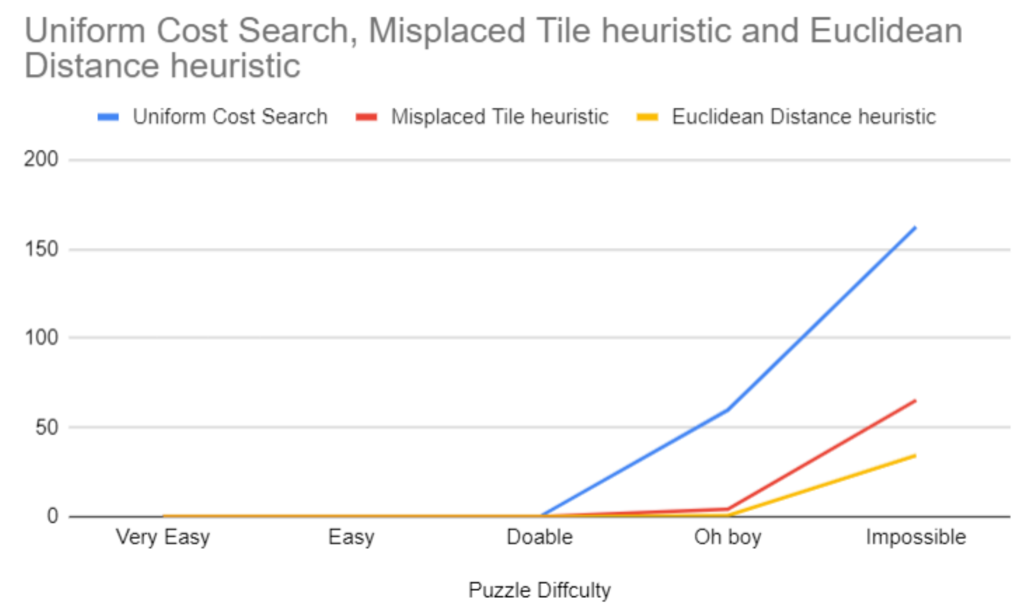
Puzzle Diffculty	Uniform Cost Search	Misplaced Tile heuristic	Euclidean Distance heuristic
Very Easy	2	1	1
Easy	6	2	2
Doable	20	4	4
Oh boy	142960	6881	1058
Impossible	500000	50697	35076

The node expanded for different puzzles is shown above. The vertical scale is 1:1000.

As we can see, as the difficulty of the puzzles increases, the number of nodes expanded increases.

The node expanded: Uniform Cost Search > Misplaced Tile heuristic > Euclidean Distance heuristic.

Maximum nodes in queue:



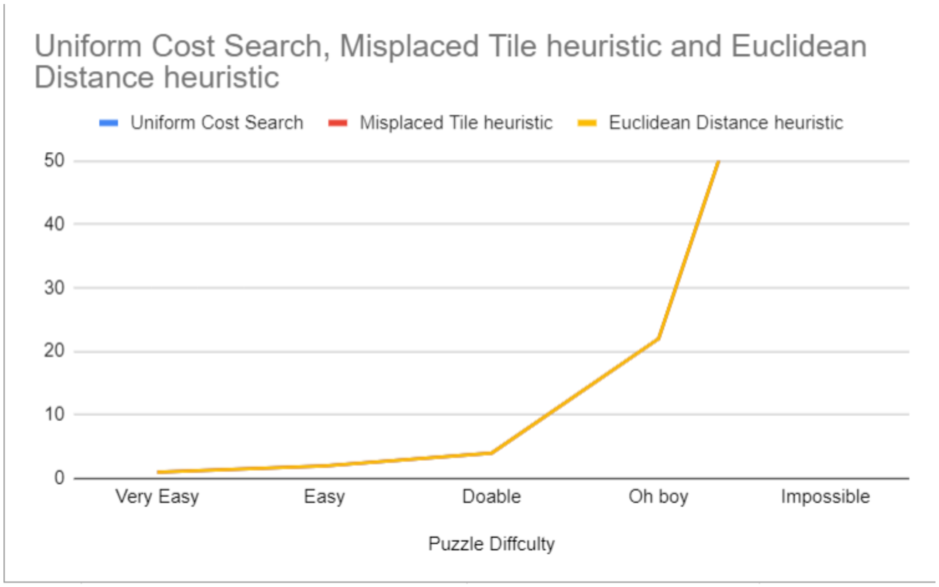
Puzzle Diffculty	Uniform Cost Search	Misplaced Tile heuristic	Euclidean Distance heuristic
Very Easy	5	3	3
Easy	8	3	3
Doable	18	4	4
Oh boy	59810	4210	610
Impossible	162424	65242	34256

The maximum nodes in the queue for different puzzles is shown above. The vertical scale is 1:1000.

As we can see, as the difficulty of the puzzles increases, the maximum nodes in the queue increases.

The maximum nodes in the queue: Uniform Cost Search > Misplaced Tile heuristic > Euclidean Distance heuristic.

Depth of the nodes:



Puzzle Diffculty	Uniform Cost Search	Misplaced Tile heuristic	Euclidean Distance heuristic
Very Easy	1	1	1
Easy	2	2	2
Doable	4	4	4
Oh boy	22	22	22
Impossible	100000	100000	100000

The depth of the nodes for different puzzles is shown above. The vertical scale is regular.

As we can see, as the difficulty of the puzzles increases, the depth of the nodes increases.

The depth of the node: Uniform Cost Search = Misplaced Tile heuristic = Euclidean Distance heuristic.

Overall, Euclidean Distance Heuristic Search is the most efficient algorithm for these three algorithms.

The difference between these three algorithms are nodes expanded and number of nodes in the queue. Euclidean Distance Heuristic Search has the least node expanded and the least number of nodes in the queue. However, no matter which algorithm we use, the depth of the nodes will remain the same for these three algorithms.