Report 8 puzzle

Challenges in this project:

Goal Configuration: Because the puzzle pieces were different sizes, figuring out the goal configuration was difficult. For each size (3x3, 4x4, and 5x5), the objective configuration was specified in order to guarantee precise comparison and termination circumstances.

Heuristic Function: The effectiveness of the search algorithm depended on the choice of the right heuristic function. Manhattan distance, Euclidean distance, and uniform cost are only a few of the heuristic functions that were taken into account. The accuracy and speed of identifying the ideal solution are impacted by the heuristic selection.

Search method: For the puzzle to be solved quickly, the right search method had to be chosen. The trade-offs between memory use and calculation time were assessed for both tree search and graph search algorithms.

Design (objects and methods):

The implementation of the Eight Puzzle solver consists of the following objects and methods:

Objects:

EightPuzzle: Represents an instance of the Eight Puzzle board. It contains the current state of the puzzle, the depth of the current state, the total cost function f, and a reference to the previous state.

goal3, goal4, goal5: Goal configurations for 3x3, 4x4, and 5x5 puzzles, respectively.

moves: Defines the possible moves for sliding tiles (up, down, left, right).

Methods:

heuristic(): Calculates the heuristic value for the current puzzle state. The implementation includes three heuristic functions: Manhattan distance, Euclidean distance, and uniform cost.

isGoal(): Checks if the current puzzle state matches the goal configuration.

neighbors(): Generates the neighboring puzzle states by sliding tiles. It returns an iterable collection of EightPuzzle objects representing the neighboring states.

printSolution(): Prints the solution path by recursively traversing the path from the goal state to the initial state.

toString(): Generates a string representation of the puzzle state, including the depth and heuristic value.

Code Optimization with Data Structures

The finding and comparison of states in the code are not optimized by using any special data structures. However, to effectively manage the puzzle states, visited states, and the search queue, it makes use of built-in Java data structures including arrays, lists, sets, and priority queues.

Findings

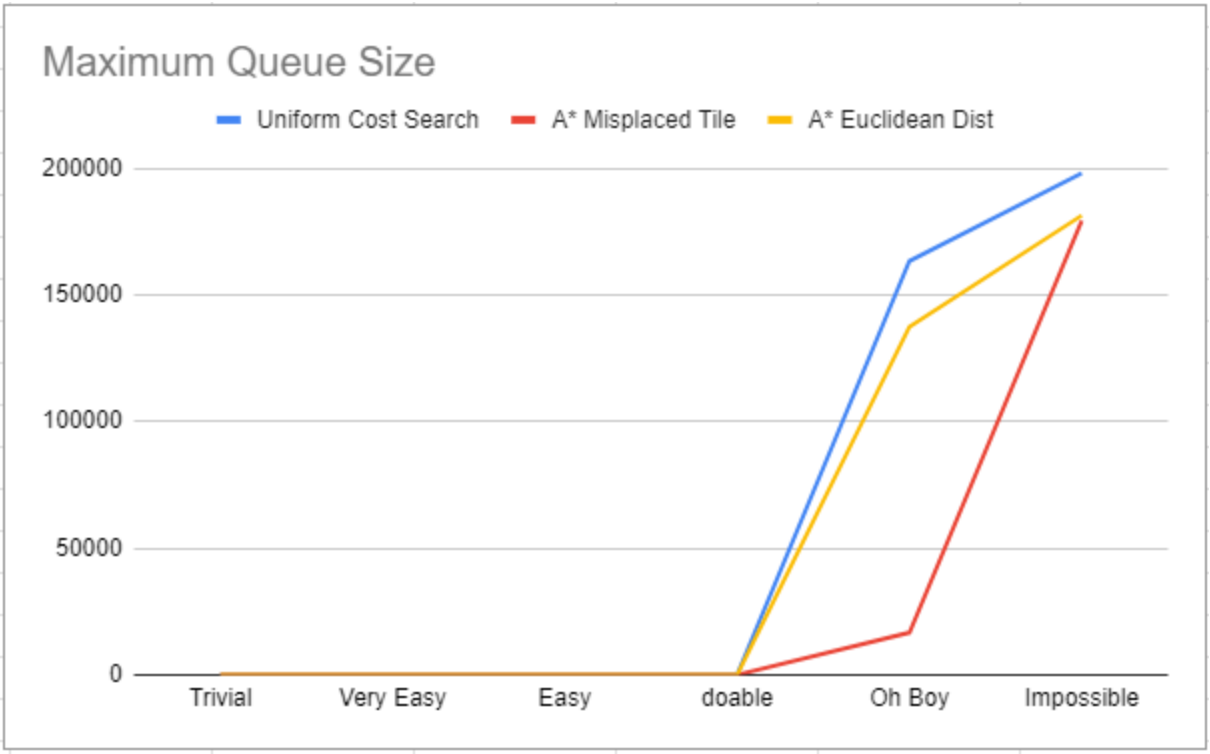
Test case :

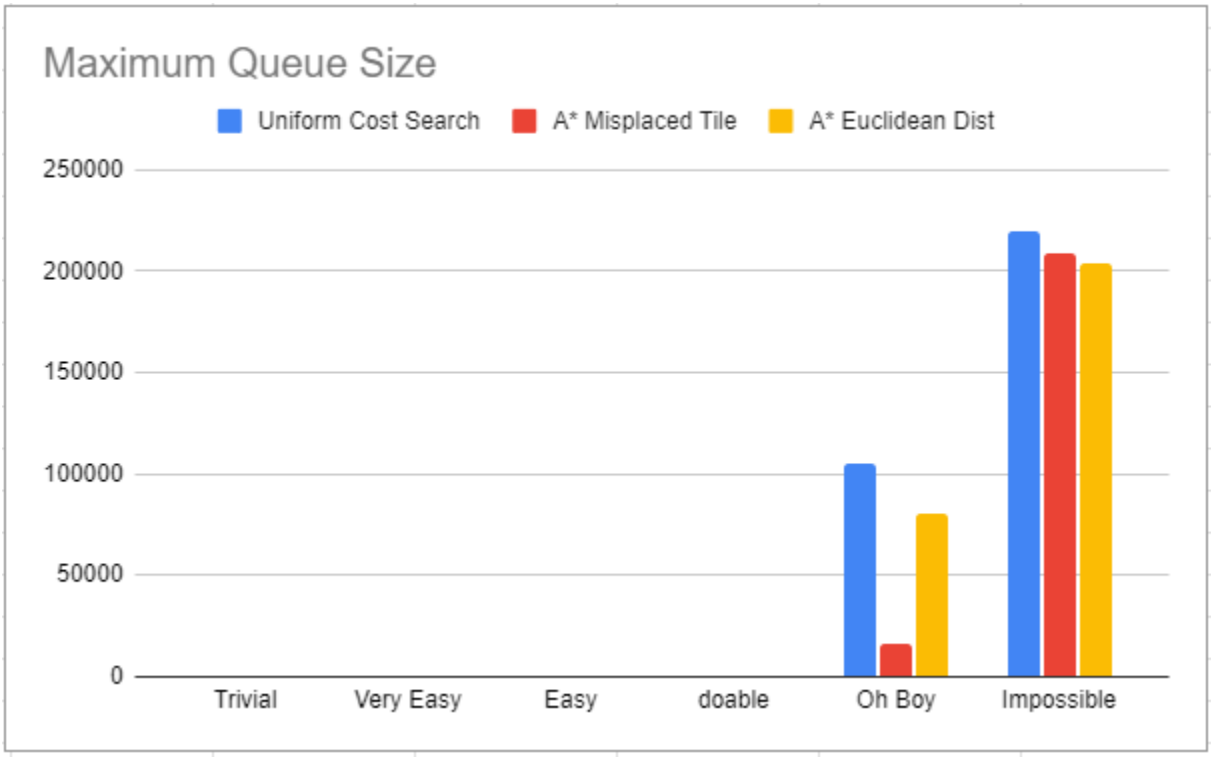
Trivial :	Very easy :	Easy :
1 2 3 4 5 6 7 8 0	1 2 3 4 5 6 7 0 8	1 2 0 4 5 3 7 8 6

Doable :	Oh Boy :	Impossible :
0 1 2	8 7 1	1 2 3
4 5 3	6 0 2	4 5 6
7 8 6	5 4 3	8 7 0

Maximum Queue Size

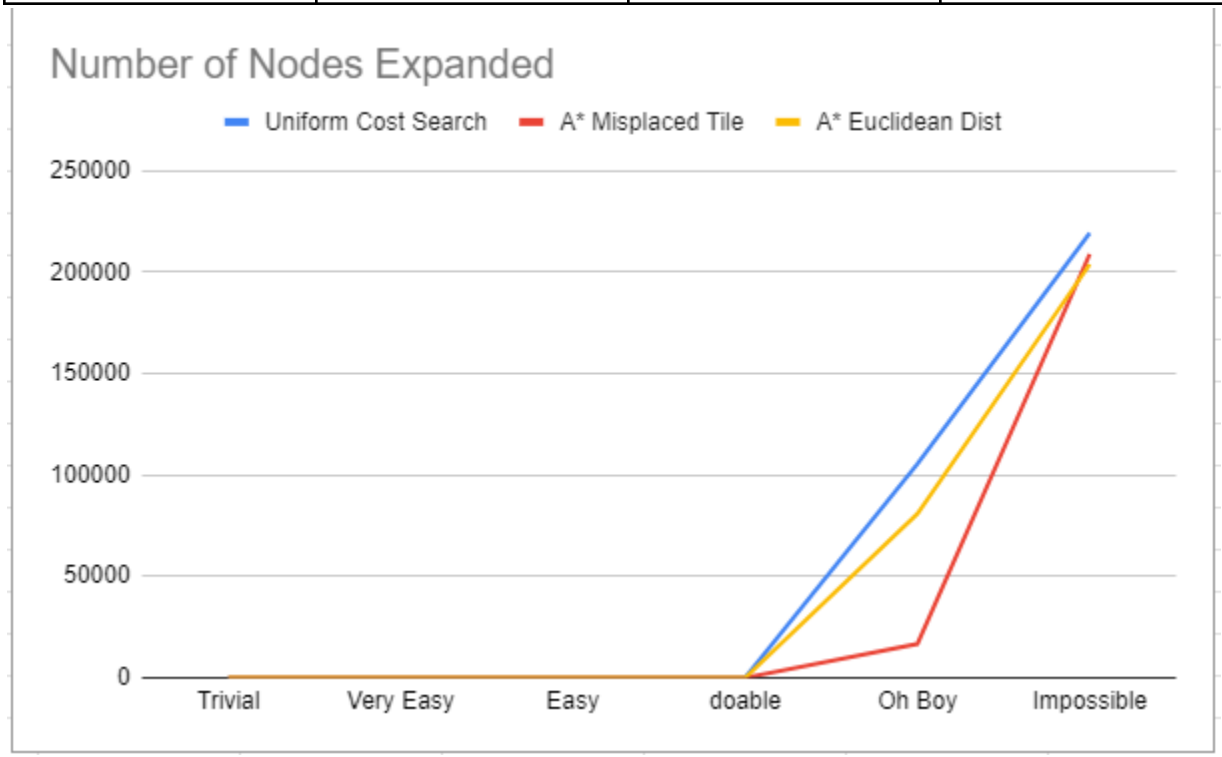
	Uniform Cost Search	A* Misplaced Tile	A* Euclidean Dist
Trivial	1	1	1
Very Easy	3	3	3
Easy	8	4	4
doable	108	7	7
Oh Boy	163497	16718	137533
Impossible	198096	179345	181439


Maximum Queue Size
Uniform Cost Search
A* Misplaced Tile
A* Euclidean Dist
200000
150000
100000
50000
0
Trivial
Very Easy
Easy
doable
Oh Boy
Impossible


Maximum Queue Size
Uniform Cost Search
A* Misplaced Tile
A* Euclidean Dist
250000
200000
150000
100000
50000
0
Trivial
Very Easy
Easy
doable
Oh Boy
Impossible

Number of Nodes Expanded

	Uniform Cost Search	A* Misplaced Tile	A* Euclidean Dist
Trivial	1	1	1
Very Easy	2	2	2
Easy	6	3	3
doable	62	5	5
Oh Boy	105432	10171	80906
Impossible	219321	208945	203852

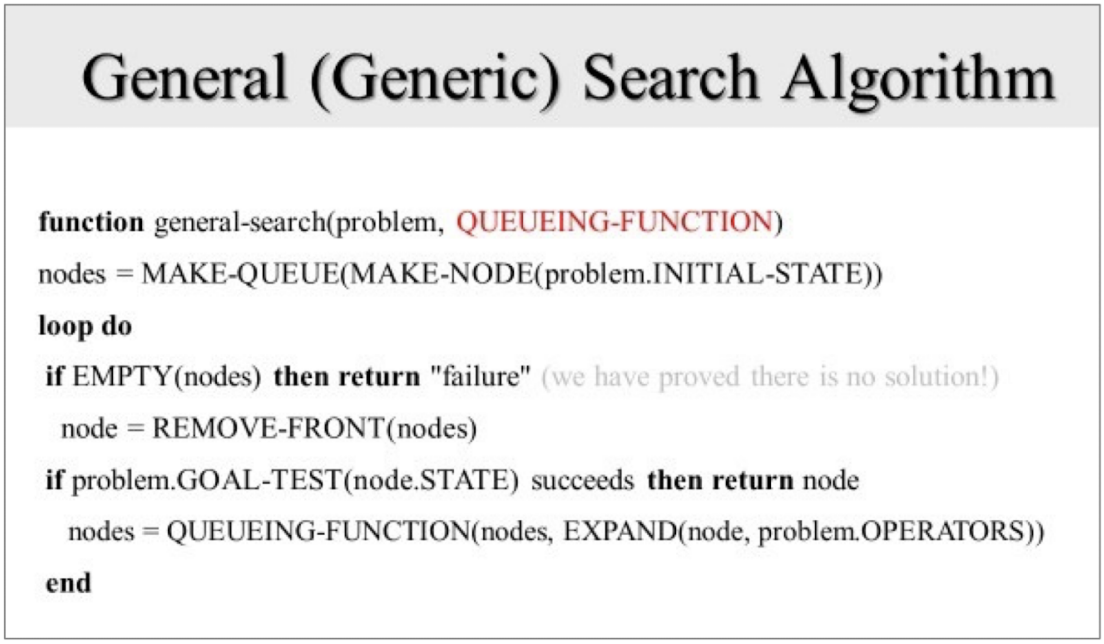


Discussion About the Results

According to our running results, the A* Misplaced Tile got the best result. Apparently, the Uniform Cost Search got the worst result. The number of the maximum queue size and the number of nodes expanded seems larger than I expected. This is probably because our code is not very efficient, we can optimize it by pruning the search tree: The search tree can be pruned by eliminating nodes that are unlikely to lead to the goal. For example, if two tiles are swapped in a move, then it is impossible to reach the goal state by swapping these tiles back individually. Therefore, these nodes can be pruned from the search tree.

Sources

- Lecture Slide: Blind Search
- Lecture Slide: Heuristic Search
- LeetCode 1162. As Far from Land as Possible
- The General Search Algorithm pseudocode from lecture:



Team Members

Siyuan Wang swang414

Pu Sun psun029

Tao Chen tchen285

Xinyu Tong xtong019