Kevin Gao SID: 862138776 kgao006@ucr.edu

CS170 Project #1: Eight Puzzle Report

Challenges

Challenges were just knowing where to start and figuring out how to structure this eight-puzzle solver. Once the structure of the project was figured out, the next biggest challenge was to code uniform-cost search. The A* misplaced tile and Euclidean searches were just variations of the original uniform cost search code, so once those 2 major challenges were solved, all that was left was to code the logic in. After struggling with figuring out the structure for some time, I talked with my partner Jordan Sam and chose to move forward with a solution containing 4 different classes, the main, a Node class, and the Problem class

The other major challenge of coding uniform cost search was somewhat mitigated by the examples from the slides and the instructions, Geeks for Geeks helped with understanding more of how the code worked together and as a baseline for some code. From those skeletons, I slowly figured out how to fit the code into the scope of our problem and add parts that were necessary. Special structures that were used were a Node class, and a priority queue to result in making a graph search.

Design

Node

Class variables: state, parentNode, cost, misplacedBy, blankRowIndex,blankColIndex
Functions: getCost, setCost, Node constructor, move, printNode
Holds different states of the problem and records parentNode, cost, and location of the blank among other variables for other classes. Allows for states to be organized and compared.

Problem

Class Variables: initial_state, goal_state, expandedCount, queueMaxCount, goalDepth, Visited
Functions: Problem constructor, isGoalState, uniformCostSearch, AStarMisplacedTile, AStarEuclidean, getExpandedCount, getQueueMaxCount, getGoalDepth, setGoalState, getGoalState, countMisplacedTiles, calculateEuclideanDistance, findNumIndexes, incrementExpanded, incrementGoalDepth, setQueueMaxCount, pushIntoVisited, nodeAlreadyVisited, leftNode, rightNode, topNode, bottomNode

Does the brunt of the work, contains the search functions for all 3 searches, as well as the operators to make different states. Each search determines what node gets priority because of a priority queue and the Priority Class combined with the costs assigned to each node based on the search desired. Searches add to and pop from the queue until the queue is empty or the goal state is found. Once a Node is chosen from the queue, it is checked to be the goal state, if it is not and the program continues, the operators run to make whatever other states(Nodes) are possible and add them to the queue while assigning the correct costs and other values needed.

Priority - Contains struct compare with a cost comparison logic line to input into priority queue

Main - Contains print statements and runs certain searches based on user input, allows user to make their own initial state and is the location where the goal state is set inside a Problem class.

Comparing Heuristic Functions -Initial State : Trivial

```
1 2 3
4 5 6
7 8 *
```

Nodes Expanded	Uniform Cost Search	Misplaced Tile	Euclidean Distance
Trivial	0	0	0
Very Easy	1	1	1
Easy	7	2	2
Doable	72	8	7

Max Nodes In Queue	Uniform Cost Search	Misplaced Tile	Euclidean Distance
Trivial	1	1	1
Very Easy	2	2	2
Easy	12	4	4
Doable	127	14	13

Time (ms)	Uniform Cost Search	Misplaced Tile	Euclidean Distance
Trivial	8	6	6
Very Easy	10	13	16
Easy	24	13	14
Doable	199	32	33