8-Puzzle

PEAS Assessment:

Performance:

- Solution found: the goal state must be reached successfully, this is the primary measure of success
- Path cost: the number of moves it took to reach the goal.
- Efficiency: the path to the goal state is the most optimal path possible

Environment:

- Fully observable: the complete state of the board is known, there is no hidden information.
- Deterministic: there is no randomness, the outcome of every action is certain.
- Sequential: each move directly impacts future moves and states.
- Static: the state only changes when an action is directly performed.
- Single-agent: there is only one agent acting within the environment.

Actuators:

- Slide blank tile up
- Slide blank tile down
- Slide blank tile right
- Slide blank tile left

Sensors:

- Will utilize 1D array: [1, 2, 3,
  4, 5, 6,
  7, 8, 0]

State-Space-Model:

- InitialState(): this function returns a random starting state that is solvable. The puzzle is solvable if the pieces are in an even permutation.
- Actions(state): given a state, this function returns all valid actions that can be applied. The actions are represented by the strings ‘n’, ‘s’, ‘e’, ‘w’ for north, south, east, and west.

- Transition(state, action): given a state and a valid action, this function returns the new state that is a result of applying said action.
- GoalTest(state): given a state, this function will return a boolean value representing whether or not it is the goal state.
- StepCost(state, action, next_state): this function will return the cost of applying an action to a given state to transition to the next state.
- Heuristic(state): given a state, this function will return the estimated cost to reach the goal from said state.

Heuristics:

h0 = 0 → UCS baseline:

- This allows the search algorithm to function as a Uniformed Cost Search, creating a baseline comparison with no actual guidance toward the goal state.

h1:

- This heuristic counts the number of tiles that are out of place when compared to the goal state. Because the number of out of place tiles would have to move at least once to get to the goal state, it is impossible for this heuristic to overestimate the true cost. Since true cost will always be equal or greater to the number of tiles that are out of place, that is proof that this is admissible.

h2:

- This is a heuristic that is stronger than the previous two because it measures how many moves out of place each tile is, rather than counting each out of place tile as 1. It does this by comparing the position of the tile to the correct position of that tile in the goal state, and tallies up the totals for each number that is not 0. It finds the minimum number of moves for each individual tile, not considering how it would affect the other tiles on the board. This creates a floor, as the total number of moves in the actual solution must be greater than or equal to this perfect minimum that we have found. Since the true cost will always be equal or greater, this is proof that this is admissible.

Experiments:

	h0	h1	h2
Average Number of Nodes Expanded	75607.57	17467.07	981.7
Average Number of Nodes Generated	89551.77	24683.87	1548.93
Average Max Frontier Size	17469.77	7199.7	562.73
Average Solution Depth/Cost	20.57	22.0	21.13

* These averages are taken from 30 instances of the domain. The values are truncated at 2 decimal places. *

Analysis:

When examining the performance of the different heuristics, it becomes clear very quickly that there is a wide gap between their efficiency levels. The baseline h0 vastly more nodes than the other two, and expanded almost all of them. The second heuristic cut down both of those metrics significantly, showing that it was much more efficient than the blind search. The third heuristic impressively brings the number of nodes generated and expanded down to just a fraction of what they were in the blind search. The number of nodes expanded went from over 75,000 to under 1000. This shows just how powerful it is at effectively pruning the search tree to remove paths that are not promising. The costs are all similar, however they do go up slightly when the heuristics are used to prune branches from the tree. It seems to be a worthwhile trade for the extreme reductions in time taken to perform the search, as using the heuristics was much more optimal. From the data pulled above, the effective branching factor can be found using the formula b* = N^(1/d), considering the number of nodes expanded and the solution depth. For h0, this is 1.73, for h1 it is 1.56, and for h2 it is 1.39. A lower number indicates a more optimal search, speaking to the effectiveness of the different heuristics discussed above.

Git Repo Link:

https://github.com/joeoneal/cs4300/tree/82114932d25ada227429cbd85db045a18d1216a6/a2