

8-Puzzle Problem Solver

The Python implementation of the 8-Puzzle Problem Solver using the A* search algorithm was provided with my answers. One tile is always missing, represented by an underscore (`_`). I have also provided details about the code through comments, but this documentation is also useful for my code readers.

The program allows the user to input the initial and goal states of the board and then uses the A* algorithm with two different heuristics - Manhattan distance and Euclidean distance - to find the optimal solution to the problem. The user can choose between these two heuristics by inputting 0 or 1, respectively.

Classes

- `Node`: A class representing each node in the search tree. Each node contains information about the current state of the board, its level in the search tree, and its calculated f-value.
- `Puzzle`: A class representing the overall puzzle problem. It contains the size of the board, the open and closed lists for the search algorithm, and the heuristics used to calculate f-values.

Methods

- `generate_child`: Generates child nodes from the given node by moving the empty space either in the four directions {up,down,left,right}.
- `shuffle`: Moves the empty space in the given direction and returns `None` if the position values are out of limits.
- `copy`: A copy function to create a similar matrix of the given node.
- `find`: Specifically used to find the position of the empty space.
- `f`: A heuristic function to calculate the f-value of a node, which is equal to the heuristic value plus the level of the node.
- `h`: A function to calculate the different between the given puzzles.
- `euclidean`: A function to calculate the Euclidean distance heuristic between the start and goal nodes.
- `manhattan`: A function to calculate the Manhattan distance heuristic between the start and goal nodes.
- `process`: Accepts the start and goal states of the board and begins the search algorithm using the chosen heuristic.

Usage

To use the program, we have to simply run the code and follow the prompts to input the initial and goal states of the board, as well as the chosen heuristic (0 for Manhattan distance, 1 for Euclidean distance). The program will then print out the optimal solution to the problem, including the number of moves required to reach the goal state and the path taken to get there.

Melvin Mokhtari (9831143) – AI – Practical Q 01 – HW 01

The screenshot shows a code editor with two panes. The left pane contains Python code for a breadth-first search algorithm to solve a 3x3 puzzle. The right pane shows the terminal output of running the script.

```
main.py
125     print(" | ")
126     print(" | ")
127     print(" \\\' / \\n")
128     for i in cur.data:
129         for j in i:
130             print(j,end=" ")
131         print("")
132     """ If the difference between current and goal node
        is 0 we have reached the goal node"""
133     if(self.h(cur.data,goal) == 0):
134         break
135     for i in cur.generate_child():
136         i.fval = self.f(i,goal)
137         self.open.append(i)
138     self.closed.append(cur)
139     del self.open[0]
140     """ sort the opne list based on f value """
141     self.open.sort(key = lambda x:x.fval,reverse=False)
142 typ = input("Enter 0 for Manhattan and 1 for Euclidean \\n")
143 puz = Puzzle(3,typ)
144 puz.process()
```

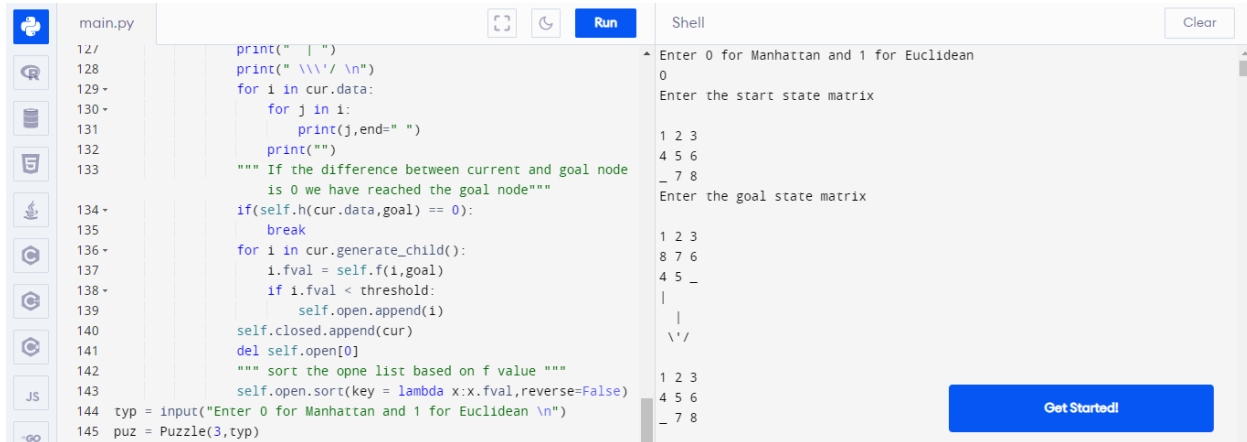
The Shell window displays the following interaction:

```
Enter 0 for Manhattan and 1 for Euclidean
1
Enter the start state matrix
1 2 4
3 5 _
7 6 8
Enter the goal state matrix
1 2 4
3 5 7
8 6 _
|
|
\\' /
1 2 4
3 5 _
7 6 8
```

A "Get Started!" button is visible at the bottom right of the interface.

Melvin Mokhtari (9831143) – AI – Practical Q 01 – HW 01

IDA* is easily created by designing a threshold for the scenario, which I also did. The threshold is a parameter in the IDA* algorithm that controls the search depth. The threshold value is set to 20 in this implementation. The algorithm will continue to look for a solution by gradually increasing the threshold until it finds one or determines that no solution exists within the given threshold limit.



```
main.py
127
128 print(" | ")
129 print(" \\\\'/ \n")
130 for i in cur.data:
131     for j in i:
132         print(j,end=" ")
133     print("")
134 """ If the difference between current and goal node
135 is 0 we have reached the goal node"""
136 if(self.h(cur.data,goal) == 0):
137     break
138 for i in cur.generate_child():
139     i.fval = self.f(i,goal)
140     if i.fval < threshold:
141         self.open.append(i)
142     self.closed.append(cur)
143     del self.open[0]
144 """ sort the opne list based on f value """
145 self.open.sort(key = lambda x:x.fval,reverse=False)
146 typ = input("Enter 0 for Manhattan and 1 for Euclidean \n")
147 puz = Puzzle(3,typ)
```

```
Shell
Enter 0 for Manhattan and 1 for Euclidean
0
Enter the start state matrix
1 2 3
4 5 6
_ 7 8
Enter the goal state matrix
1 2 3
8 7 6
4 5 _
|
|
\'/
1 2 3
4 5 6
_ 7 8
```

Get Started!