Assignment: 8 Puzzle Game

CSE-0408 Summer 2021

Afifa Nowreen Akhter
Department of Computer Science and Engineering
State University of Bangladesh (SUB)
Dhaka, Bangladesh
afifa.nowreen@gmail.com

***Abstract*—The puzzle can be solved by moving the tiles one by one in the single empty space and thus achieving the Goal state. Instead of moving the tiles in the empty space we can visualize moving the empty space in place of the tile. The empty space cannot move diagonally and can take only one step at a time.**

***Index Terms*—Puzzle, Tiles**

I. Introduction

The 8-puzzle problem is a puzzle invented and popularized by Noyes Palmer Chapman in the 1870s. It is played on a 3-by-3 grid with 8 square blocks labeled 1 through 8 and a blank square. Your goal is to rearrange the blocks so that they are in order.

II. Literature Review

Sadikov and Bratko (2006) studied the suitability of pessimistic and optimistic heuristic functions for a real-time search in the 8-puzzle. They discovered that pessimistic functions are more suitable. They also observed the pathology, which was stronger with the pessimistic heuristic function. However, they did not study the influence of other factors on the pathology or provide any analysis of the gain of a deeper search.

III. Conclusion

I test my code to see how many states it would take to get from the current state to the goal state. Then it came up with three.

Acknowledgment

I would like to thank my honourable **Khan Md. Hasib Sir** for his time, generosity and critical insights into this project.

References

[1] Piltaver, R., Lustrek, M., and Gams, M. (2012). The pathology of heuristic ˇ search in the 8-puzzle. Journal of Experimental and Theoretical Artificial Intelligence, 24(1), 65-94.

[2] Igwe, K., Pillay, N., Rae, C. (2013, October). Solving the 8-puzzle problem using genetic programming. In Proceedings of the South African Institute for Computer Scientists and Information Technologists Conference (pp. 64-67).

[3] Reinefeld, A. (1993, August). Complete Solution of the Eight-Puzzle and the Bene t of Node Ordering in IDA*. In International Joint Conference on Artificial Intelligence (pp. 248-253).

[4]] Shaban, R. Z., Natheer Alkallak, I., Mohamad Sulaiman, M. (2010). Genetic Algorithm to Solve Sliding Tile 8-Puzzle Problem. Journal of Education and Science, 23(3), 145-157.

[5] Nayak, D. (2014). Analysis and Implementation of Admissible Heuristics in 8 Puzzle Problem (Doctoral dissertation)

[6] Mishra, A. K., Siddalingaswamy, P. C. (2017, April). Analysis of tree based search techniques for solving 8-puzzle problem. In 2017 Innovations in Power and Advanced Computing Technologies (i-PACT) (pp. 1-5). IEEE.

```python
from copy import deepcopy
from colorama import Fore, Back, Style

DIRECTIONS = {"D": [-1, 0], "U": [1, 0], "R": [0, -1], "L": [0, 1]}
END = [[1, 2, 3], [8, 0, 4], [7, 6, 5]]

# unicode
left_down_angle = '\u2514'
right_down_angle = '\u2518'
right_up_angle = '\u2510'
left_up_angle = '\u250C'

middle_junction = '\u253C'
top_junction = '\u252C'
bottom_junction = '\u2534'
right_junction = '\u2524'
left_junction = '\u251C'

bar = Style.BRIGHT + Fore.CYAN + '\u2502' + Fore.RESET + Style.RESET_ALL
dash = '\u2500'

first_line = Style.BRIGHT + Fore.CYAN + left_up_angle + dash + dash + dash + top_junction + dash + dash + dash + top_junction + c
middle_line = Style.BRIGHT + Fore.CYAN + left_junction + dash + dash + dash + middle_junction + dash + dash + dash + middle_junct
last_line = Style.BRIGHT + Fore.CYAN + left_down_angle + dash + dash + dash + bottom_junction + dash + dash + dash + bottom_junct


def print_puzzle(array):
    print(first_line)
    for a in range(len(array)):
        for i in array[a]:
            if i == 0:
```

Fig. 1. Code

```python
                print(bar, Back.RED + ' ' + Back.RESET, end=' ')
            else:
                print(bar, i, end=' ')
        print(bar)
        if a == 2:
            print(last_line)
        else:
            print(middle_line)


class Node:
    def __init__(self, current_node, previous_node, g, h, dir):
        self.current_node = current_node
        self.previous_node = previous_node
        self.g = g
        self.h = h
        self.dir = dir

    def f(self):
        return self.g + self.h


def get_pos(current_state, element):
    for row in range(len(current_state)):
        if element in current_state[row]:
            return (row, current_state[row].index(element))


def euclidianCost(current_state):
    cost = 0
    for row in range(len(current_state)):
        for col in range(len(current_state[0])):
```

Fig. 2. Code

```python
    for row in range(len(current_state)):
        for col in range(len(current_state[0])):
            pos = get_pos(END, current_state[row][col])
            cost += abs(row - pos[0]) + abs(col - pos[1])
    return cost


def getAdjNode(node):
    listNode = []
    emptyPos = get_pos(node.current_node, 0)

    for dir in DIRECTIONS.keys():
        newPos = (emptyPos[0] + DIRECTIONS[dir][0], emptyPos[1] + DIRECTIONS[dir][1])
        if 0 <= newPos[0] < len(node.current_node) and 0 <= newPos[1] < len(node.current_node[0]):
            newState = deepcopy(node.current_node)
            newState[emptyPos[0]][emptyPos[1]] = node.current_node[newPos[0]][newPos[1]]
            newState[newPos[0]][newPos[1]] = 0
            # listNode += [Node(newState, node.current_node, node.g + 1, euclidianCost(newState), dir)]
            listNode.append(Node(newState, node.current_node, node.g + 1, euclidianCost(newState), dir))

    return listNode


def getBestNode(openSet):
    firstIter = True

    for node in openSet.values():
        if firstIter or node.f() < bestF:
            firstIter = False
            bestNode = node
            bestF = bestNode.f()
    return bestNode
```

Fig. 3. Code

```python
def buildPath(closedSet):
    node = closedSet[str(END)]
    branch = list()

    while node.dir:
        branch.append({
            'dir': node.dir,
            'node': node.current_node
        })
        node = closedSet[str(node.previous_node)]
    branch.append({
        'dir': '',
        'node': node.current_node
    })
    branch.reverse()

    return branch


def main(puzzle):
    open_set = {str(puzzle): Node(puzzle, puzzle, 0, euclidianCost(puzzle), "")}
    closed_set = {}

    while True:
        test_node = getBestNode(open_set)
        closed_set[str(test_node.current_node)] = test_node

        if test_node.current_node == END:
            return buildPath(closed_set)

        adj_node = getAdjNode(test_node)
        for node in adj_node:
```

Fig. 4. Code

```python
                str(node.current_node)].f() < node.f():
                continue
            open_set[str(node.current_node)] = node

        del open_set[str(test_node.current_node)]


if __name__ == '__main__':
    br = main([[1, 2, 3],
               [8, 6, 0],
               [7, 5, 4]])

    print('total steps : ', len(br) - 1)
    print()
    print(dash + dash + right_junction, "INPUT", left_junction + dash + dash)
    for b in br:
        if b['dir'] != '':
            letter = ''
            if b['dir'] == 'U':
                letter = 'UP'
            elif b['dir'] == 'R':
                letter = "RIGHT"
            elif b['dir'] == 'L':
                letter = 'LEFT'
            elif b['dir'] == 'D':
                letter = 'DOWN'
            print(dash + dash + right_junction, letter, left_junction + dash + dash)
        print_puzzle(b['node'])
        print()

    print(dash + dash + right_junction, 'ABOVE IS THE OUTPUT', left_junction + dash + dash)
```

Fig. 5. Code