Astar Assignment (Project 1): Nicholas Cassarino

Github Link: https://github.com/nmcassa/Astar_assignment

Problem Formulation:

The 8-puzzle problem is a game played on a 3 x 3 board with 8 tiles and one one square. The goal of the game is to arrange the blocks in correspondence to a goal state that is provided when you start the game (as shown below). You are to take sequential steps in order to achieve this goal. The difficulty comes from planning moves in advance and choosing the correct order to move these tiles around.

In our case, we will be using a path finding algorithm in order to solve given input and goal states. We will be using Astar search which is an algorithm that uses the path up until this point and a heuristic score in order to make decisions in each state. The heuristic we will be using is Manhattan distance, more specifically the distance that all tiles would need to move in order to achieve the goal state.

Program Structure (variables, functions and procedures):

For starters I wrote a simple function (*read_two_grids*) that reads the initial state and goal state from a text file. In my github repo (https://github.com/nmcassa/Astar_assignment), all inputs can be found within the input directory. README.md file shows how to run things, and there are no requirements to run the python file.

Astar is an iterative algorithm that iterates through states as they are expanded. Therefore, within the main function we call a function, *A_star_iter*, iteratively until we find a solution or we decide that we cannot find a solution. Each iteration, we choose a node to expand.

Within our iteration function we first generate all possible states from our current state (*make_future_states*). We do this by finding the empty space within our state (0) and find all the possible swaps/movements that we can make around it.

Then, we take those states and compute the heuristic scores for all of them. We do this by iterating through the possible states (*compute_next_state*) and computing their Manhattan distances against the goal state (*compute_score*). After computing each score we test to see if it is the best/lowest score so far, we later return the best scoring state as our next state to expand.

Once we have found a state that equals our goal state or we have decided that we cannot find a solution, we are done. We use one global variable that tracks the total amount of generated nodes.

Analyze Input/Output Cases:

- Basic.txt
    - Generated nodes: 24
    - Expanded nodes: 8
    - We do reach a solution
    - Path below

Input:
[1, 2, 3]
[7, 4, 5]
[6, 8, 0]

Goal:
[1, 2, 3]
[8, 6, 4]
[7, 5, 0]

Step #1
[1, 2, 3]
[7, 4, 0]
[6, 8, 5]

Step #2
[1, 2, 3]
[7, 0, 4]
[6, 8, 5]

Step #3
[1, 2, 3]
[7, 8, 4]
[6, 0, 5]

Step #4
[1, 2, 3]
[7, 8, 4]
[0, 6, 5]

Step #5
[1, 2, 3]
[0, 8, 4]
[7, 6, 5]

Step #6
[1, 2, 3]
[8, 0, 4]
[7, 6, 5]

Step #7
[1, 2, 3]
[8, 6, 4]
[7, 0, 5]

Step #8
[1, 2, 3]
[8, 6, 4]

[7, 5, 0]

- Basic1.txt
    - Generated nodes: 17
    - Expanded nodes: 6
    - We do reach a solution
    - Path below

Input:
[2, 8, 1]
[3, 4, 6]
[7, 5, 0]

Goal:
[3, 2, 1]
[8, 0, 4]
[7, 5, 6]

Step #1
[2, 8, 1]
[3, 4, 0]
[7, 5, 6]

Step #2
[2, 8, 1]
[3, 0, 4]
[7, 5, 6]

Step #3
[2, 0, 1]
[3, 8, 4]
[7, 5, 6]

Step #4
[0, 2, 1]

[3, 8, 4]
[7, 5, 6]

Step #5
[3, 2, 1]
[0, 8, 4]
[7, 5, 6]

Step #6
[3, 2, 1]
[8, 0, 4]
[7, 5, 6]

- Basic2.txt
    - Generated nodes: 252
    - Expanded nodes: 100
    - We do not reach a solution
        - Loop
    - Loop shown below
    - Distance from goal: 12

Input:
[7, 2, 4]
[5, 0, 6]
[8, 3, 1]

Goal:
[1, 2, 3]
[4, 5, 6]
[7, 8, 0]

....

Step #93
[7, 2, 4]

[0, 3, 6]
[5, 8, 1]

Step #94
[0, 2, 4]
[7, 3, 6]
[5, 8, 1]

Step #95
[7, 2, 4]
[0, 3, 6]
[5, 8, 1]

Step #96
[0, 2, 4]
[7, 3, 6]
[5, 8, 1]

- random.txt
    - Generated nodes: 12
    - Expanded nodes: 4
    - We do reach a solution
    - Path shown below

Input:
[0, 1, 3]
[4, 2, 5]
[7, 8, 6]

Goal:
[1, 2, 3]
[4, 5, 6]
[7, 8, 0]

Step #1
[1, 0, 3]

[4, 2, 5]
[7, 8, 6]

Step #2
[1, 2, 3]
[4, 0, 5]
[7, 8, 6]

Step #3
[1, 2, 3]
[4, 5, 0]
[7, 8, 6]

Step #4
[1, 2, 3]
[4, 5, 6]
[7, 8, 0]

- random2.txt
    - Generated nodes: 250
    - Expanded nodes: 100
    - We do not reach a solution
        - Loop
    - Loop shown below
    - Distance from goal: 13

Input:
[4, 5, 6]
[2, 8, 7]
[1, 3, 0]

Goal:
[1, 2, 3]
[2, 5, 8]
[4, 6, 0]

....

Step #95
[4, 5, 6]
[2, 8, 0]
[1, 3, 7]

Step #96
[4, 5, 0]
[2, 8, 6]
[1, 3, 7]

Step #97
[4, 5, 6]
[2, 8, 0]
[1, 3, 7]

Step #98
[4, 5, 0]
[2, 8, 6]
[1, 3, 7]

- random3.txt
    - Generated nodes: 250
    - Expanded nodes: 100
    - We do not reach a solution
        - Loop
    - Loop shown below
    - Distance from goal: 5

Input:
[1, 4, 5]
[0, 8, 7]
[6, 3, 2]

Goal:
[0, 1, 4]
[5, 8, 7]
[6, 3, 2]

….

Step #97
[0, 4, 5]
[1, 8, 7]
[6, 3, 2]

Step #98
[1, 4, 5]
[0, 8, 7]
[6, 3, 2]

Step #99
[0, 4, 5]
[1, 8, 7]
[6, 3, 2]

Step #100
[1, 4, 5]
[0, 8, 7]
[6, 3, 2]

- random4.txt
    - Generated nodes: 250
    - Expanded nodes: 100
    - We do not reach a solution
        - Loop
    - Loop shown below

    - Distance from goal: 3

Input:
[1, 2, 3]
[4, 6, 0]
[7, 8, 5]

Goal:
[1, 2, 3]
[4, 0, 5]
[7, 8, 6]

....

Step #97
[1, 2, 3]
[4, 6, 5]
[7, 8, 0]

Step #98
[1, 2, 3]
[4, 6, 0]
[7, 8, 5]

Step #99
[1, 2, 3]
[4, 6, 5]
[7, 8, 0]

Step #100
[1, 2, 3]
[4, 6, 0]
[7, 8, 5]

- random5.txt

- Generated nodes: 17
- Expanded nodes: 7
- We do reach a solution
- Path shown below

Input:
[1, 2, 3]
[4, 5, 6]
[7, 8, 0]

Goal:
[2, 3, 6]
[1, 5, 0]
[4, 7, 8]

Step #1
[1, 2, 3]
[4, 5, 6]
[7, 0, 8]

Step #2
[1, 2, 3]
[4, 5, 6]
[0, 7, 8]

Step #3
[1, 2, 3]
[0, 5, 6]
[4, 7, 8]

Step #4
[0, 2, 3]
[1, 5, 6]
[4, 7, 8]

Step #5
[2, 0, 3]
[1, 5, 6]
[4, 7, 8]

Step #6
[2, 3, 0]
[1, 5, 6]
[4, 7, 8]

Step #7
[2, 3, 6]
[1, 5, 0]
[4, 7, 8]