8 Puzzle Solver using A star algorithm

Implement by : Anup Bharade and Sumedh Joglekar
Language : Python
Data Structure Used : Priority Queue

- **Puzzle formulation :**
  This program takes input and goal state from user in a 1D array as below numbers from 0 to 8.
  [1,2,3,4,5,6,7,8,0]
  Input and goal states entered are then converted to puzzle form using convert function as below:

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

  Also user must enter heuristic type as ‘0 for Manhattan and 1 for Misplaced’.

- **Program Structure :**
  Functions and Class:
  - Class Puzzle
    Global variables of class:
    heuristic : Heuristic function value to None
    f : f(n) initialized to None
    num_of_instances : To calculate the number of nodes generated
    Class puzzle takes five arguments as:

    State : Input state
    Parent : To keep track of parent’s g(n) value
    Action : This is additional variable to find solution path in reverse order from find_solution.
    g : g(n) path cost from the parent
    heuristic_type: to check which heuristic to choose from user’s input.

    Methods in class Puzzle:
    manhattan_heuristic : This method calculates the manhattan heuristic (h) using the index of the numbers.

    misplaced_heuristic : This method calculates the misplaced heuristic (h) using the index of numbers

    goal_test : This method checks whether goal state is reached or not.

    Find_Valid_actions: This is static method which decides the legal actions allowed for the index of 0 (which is null) passed from generate_successor method

    generate successor : This method is used to create successors based on valid actions and stores the successors in an array.

    find_solution: A method to find the solution path which can be accessed in reverse order to find the depth of the path.

Astar_search:

This is main method in which astar algorithm logic is implemented. Method takes I initial_state and heuristic_type as arguments.

Algorithm for Astar:

1. Initialize explored list.
2. Define start_node by getting values from Puzzle class.
3. Initialize priority queue.
4. Insert start_node to priority queue.
5. While the q is not empty
6. {
7. Get the node from priority queue
8. Add it in explored list
9. if node == goal print goal state found also print number of steps expanded
10. else generate successors from generate_successors method.
11. for every successor generated
12. {
13. if successor is not explored put it in priority queue
14. }
15. }
16. Return

Sample implementation:

1. Take input state, goal state and heuristic type as 0 or 1 from user
   For example
   input_state = [0 1 3 4 2 5 7 8 6]
   goal_state = [1 2 3 7 4 5 6 8 0]
   heuristic type = 0 (manhattan)
2. Call to Astar_search function
   - Start_node = Solver([0 1 3 4 2 5 7 8 6],None,None,0,0) # call to Solver function
   - Calculate g of parent = 0
   - if heuristic_type = 0 calculate manhattan distance
     and calculate value of f = 0+ 4
   - put start_node and value of f to priority queue
   - while queue is not empty
   - node = [0 1 3 4 2 5 7 8 6]
   - add node to explored array
   - if goal_test is achieved print('Goal !'), print('Number of nodes expanded:')
   - else:
   - generate successors as [4 1 3 0 2 5 7 8 6] and [1 0 3 4 2 5 7 8 6]
   - if successor not in explored
     put in queue.
   - return