Alex Shpyrko

Instructions: The python file has all of the code in one place. In order to run A* search on a puzzle, pass the input filename (which is in the same directory as puzzle.py) into the Fifteen_Puzzle_ASearch class when it is instantiated. Then call the execute() method to generate an output file with the solution.

Source Code:

```python
'''
The State class holds a parent value with a reference to it's parent state, a puzzle value which is a 2D array
representation of the puzzle, a depth value for the current depth, and the state's f(n) value.
'''
class State:
    def __init__(self, parent, puzzle, depth, fn):
        self.parent = parent
        self.puzzle = puzzle
        self.depth = depth
        self.fn = fn

    # Find's the position of the 0
    def find_open_space(self, curr_puzzle):
        for x in range(4):
            for y in range(4):
                if curr_puzzle[x][y] == 0:
                    return x, y

    # Swaps two values and returns the new puzzle
    def create_new_puzzle(self, x_original, y_original, x_new, y_new):
        if (x_new >= 0 and x_new < 4 and y_new >= 0 and y_new < 4):
            new_puzzle = self.copy_puzzle(self.puzzle)
            save_val = new_puzzle[x_new][y_new]
            new_puzzle[x_new][y_new] = new_puzzle[x_original][y_original]
            new_puzzle[x_original][y_original] = save_val
            return new_puzzle
        else:
            return None

    # Deep copy of puzzle
    def copy_puzzle(self, state):
        puzz = []
        for i in state:
            p = []
            for j in i:
                p.append(j)
            puzz.append(p)
        return puzz

    # Creates new children states to continue search
    def explore_children(self):
        children = []
        x_coordinate, y_coordinate = self.find_open_space(self.puzzle)
        possible_children = [[x_coordinate, y_coordinate+1], [x_coordinate,
y_coordinate-1],
                             [x_coordinate+1, y_coordinate], [x_coordinate-1,
y_coordinate]]
        for i in possible_children:
            new_puzzle = self.create_new_puzzle(x_coordinate, y_coordinate, i[0], 
i[1])
            if new_puzzle is not None:
                child = State(self, new_puzzle, self.depth+1, 0)
                children.append(child)
        return children

"""
The puzzle is solved through the Fifteen_Puzzle_ASearch class which takes an 
input_file name, stores the explored states,
and stores unexplored children
"""
class Fifteen_Puzzle_ASearch:

    def __init__(self, input_file):
        self.input_file = input_file
        self.unexplored = []
        self.explored = []

    # Calculates f(n) of the state with it's depth as g(n) and using calculate_hn to 
get a h(n) value
    def calculate_fn(self, state, goal):
        return self.calculate_hn(state.puzzle,goal) + state.depth

    # Calculates manhattan sum of the puzzle representation
    def calculate_hn(self, start_puzzle, goal_puzzle):
        manhattan_sum = 0
        for i in range(3):
            for j in range(3):
                if (start_puzzle[i][j] != goal_puzzle[i][j]):
                    manhattan_sum += 1
        return manhattan_sum

    # Uses the array of states in the solution to return the {L, R, U, D} 
representation of the solution's moves
    def calculate_moves(self, solution):
        solution.reverse()
        moves = []
        for i in range(len(solution)-1):
            first_x, first_y = solution[i].find_open_space(solution[i].puzzle)
            second_x, second_y = solution[i+1].find_open_space(solution[i+1].puzzle)
            if (first_x == second_x):
                result = first_y - second_y
                if (result > 0):
                    moves.append("L")
                else:
                    moves.append("R")
            else:
                result = first_x - second_x
                if (result > 0):
                    moves.append("U")
                else:
                    moves.append("D")
        return moves

    # Calculates the f(n) values of the states in the solution
    def solution_fn(self, solution):
        solution.reverse()
        moves_fn = []
```

```python
        for i in solution:
            moves_fn.append(i.fn)
        return moves_fn

    # Parses input file to get the start puzzle and the goal puzzle
    def get_puzzles(self, filename):
        f = open(filename, "r")
        start = []
        end = []
        for i in range(4):
            str_row = f.readline().split()
            num_row = []
            for num in str_row:
                num_row.append(int(num))
            start.append(num_row)
        f.readline()
        for i in range(4):
            str_row = f.readline().split()
            num_row = []
            for num in str_row:
                num_row.append(int(num))
            end.append(num_row)

        f.close()
        return start, end

    def generate_ouput(self, start, end, depth, n_nodes, solution_moves, fn_moves):
        name = "Output" + self.input_file[5] + ".txt"
        output_file = open(name, "w")
        for row in start:
            for num in row:
                output_file.write(str(num) + " ")
            output_file.write("\n")
        output_file.write("\n")
        for row in end:
            for num in row:
                output_file.write(str(num) + " ")
            output_file.write("\n")
        output_file.write("\n")
        output_file.write(str(depth) + "\n")
        output_file.write(str(n_nodes) + "\n")
        output_file.write(" ".join(solution_moves) + "\n")
        for num in fn_moves:
            output_file.write(str(num) + " ")

    # Execute A* Search and creates output file with solution
    def execute(self):
        start, end = self.get_puzzles(self.input_file)
        solution = []
        n_nodes = 0

        root = State(None, start, 0, -1)
        root.fn = self.calculate_fn(root, end)
        self.unexplored.append(root)
        while True:
            curr_state = self.unexplored[0]
            if (self.calculate_hn(curr_state.puzzle, end) == 0):
                node = curr_state
                while (node != None):
                    solution.append(node)
                    print(node.puzzle)
                    node = node.parent
                output_depth = curr_state.depth
                solution_moves = self.calculate_moves(solution)
                solution_fn = self.solution_fn(solution)
                self.generate_ouput(start, end, output_depth, n_nodes, solution_moves,
solution_fn)
                break
            for state in curr_state.explore_children():
                if state not in self.explored:
                    state.fn = self.calculate_fn(state, end)
                    self.unexplored.append(state)
            self.explored.append(curr_state)
            n_nodes += 1
            del self.unexplored[0]

            self.unexplored.sort(key=lambda x: x.fn, reverse=False)

input1 = Fifteen_Puzzle_ASearch("Input1.txt")
input2 = Fifteen_Puzzle_ASearch("Input2.txt")
input3 = Fifteen_Puzzle_ASearch("Input3.txt")
input4 = Fifteen_Puzzle_ASearch("Input4.txt")
input1.execute()
input2.execute()
input3.execute()
input4.execute()
```

Output1.txt:

```
1 2 3 4
5 6 0 7
8 9 10 11
12 13 14 15

1 2 3 4
5 9 6 7
8 13 0 11
12 14 10 15

5
9
L D D R U
5 5 4 4 4 4
```

Output2.txt:

```
1 5 3 13
8 0 6 4
15 10 7 9
11 14 2 12

1 5 3 13
8 10 6 4
0 15 2 9
11 7 14 12

6
68
D R D L U L
6 7 6 5 5 4 4
```

Output3.txt:

```
9 13 7 4
12 3 0 1
2 15 5 6
14 10 11 8

13 3 7 4
9 1 0 6
12 2 5 8
14 15 10 11

12
3005
R D D L L U L U U R D R
12 13 13 13 13 13 13 12 11 10 9 8 6
```

Output4.txt:

```
13 12 2 11
10 1 8 9
0 3 15 14
6 4 7 5

10 13 12 11
8 1 2 9
3 4 15 5
6 0 14 7

10
82
R U R U L L D R D D
10 10 10 10 10 10 10 10 9 7 7
```