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:

*'''  
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 = []  
 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