|  |  |
| --- | --- |
|  |  |

Problem description:

For this assignment, you will Implement A\* algorithm for the following problem:

* 8 Puzzle

Code:

import random  
import math  
  
\_goal\_state = [[0, 1, 2],  
 [3, 4, 5],  
 [6, 7, 8]]  
  
  
def index(item, seq):  
 if item in seq:  
 return seq.index(item)  
 else:  
 return -1  
  
  
class EightPuzzle:  
  
 def \_\_init\_\_(self):  
 self.\_hval = 0  
  
 self.\_depth = 0  
  
 self.\_parent = None  
 self.adj\_matrix = []  
 for i in range(3):  
 self.adj\_matrix.append(\_goal\_state[i][:])  
  
 def \_\_eq\_\_(self, other):  
 if self.\_\_class\_\_ != other.\_\_class\_\_:  
 return False  
 else:  
 return self.adj\_matrix == other.adj\_matrix  
  
 def \_\_str\_\_(self):  
 res = ''  
 for row in range(3):  
 res += ' '.join(map(str, self.adj\_matrix[row]))  
 res += '\r\n'  
 return res  
  
 def \_clone(self):  
 p = EightPuzzle()  
 for i in range(3):  
 p.adj\_matrix[i] = self.adj\_matrix[i][:]  
 return p  
  
 def \_get\_legal\_moves(self):  
 row, col = self.find(0)  
 free = []  
 if row > 0:  
 free.append((row - 1, col))  
 if col > 0:  
 free.append((row, col - 1))  
 if row < 2:  
 free.append((row + 1, col))  
 if col < 2:  
 free.append((row, col + 1))  
  
 return free  
  
 def \_generate\_moves(self):  
 free = self.\_get\_legal\_moves()  
 zero = self.find(0)  
  
 def swap\_and\_clone(a, b):  
 p = self.\_clone()  
 p.swap(a, b)  
 p.\_depth = self.\_depth + 1  
 p.\_parent = self  
 return p  
  
 return map(lambda pair: swap\_and\_clone(zero, pair), free)  
  
 def \_generate\_solution\_path(self, path):  
 if self.\_parent == None:  
 return path  
 else:  
 path.append(self)  
 return self.\_parent.\_generate\_solution\_path(path)  
  
 def set(self, other):  
 i = 0;  
 for row in range(3):  
 for col in range(3):  
 self.adj\_matrix[row][col] = int(other[i])  
 i = i + 1  
  
 def solve(self, h):  
  
 def is\_solved(puzzle):  
 return puzzle.adj\_matrix == \_goal\_state  
  
 openl = [self]  
 closedl = []  
 move\_count = 0  
 while len(openl) > 0:  
 x = openl.pop(0)  
 move\_count += 1  
 if (is\_solved(x)):  
 if len(closedl) > 0:  
 return x.\_generate\_solution\_path([]), move\_count  
 else:  
 return [x]  
  
 succ = x.\_generate\_moves()  
 idx\_open = idx\_closed = -1  
 for move in succ:  
 *# have we already seen this node?* idx\_open = index(move, openl)  
 idx\_closed = index(move, closedl)  
 hval = h(move)  
 fval = hval + move.\_depth  
  
 if idx\_closed == -1 and idx\_open == -1:  
 move.\_hval = hval  
 openl.append(move)  
 elif idx\_open > -1:  
 copy = openl[idx\_open]  
 if fval < copy.\_hval + copy.\_depth:  
 *# copy move's values over existing* copy.\_hval = hval  
 copy.\_parent = move.\_parent  
 copy.\_depth = move.\_depth  
 elif idx\_closed > -1:  
 copy = closedl[idx\_closed]  
 if fval < copy.\_hval + copy.\_depth:  
 move.\_hval = hval  
 closedl.remove(copy)  
 openl.append(move)  
  
 closedl.append(x)  
 openl = sorted(openl, key=lambda p: p.\_hval + p.\_depth)  
  
 *# if finished state not found, return failure* return [], 0  
  
 def shuffle(self, step\_count):  
 for i in range(step\_count):  
 row, col = self.find(0)  
 free = self.\_get\_legal\_moves()  
 target = random.choice(free)  
 self.swap((row, col), target)  
 row, col = target  
  
 def find(self, value):  
 if value < 0 or value > 8:  
 raise Exception("value out of range")  
  
 for row in range(3):  
 for col in range(3):  
 if self.adj\_matrix[row][col] == value:  
 return row, col  
  
 def peek(self, row, col):  
 return self.adj\_matrix[row][col]  
  
 def poke(self, row, col, value):  
 self.adj\_matrix[row][col] = value  
  
 def swap(self, pos\_a, pos\_b):  
 temp = self.peek(\*pos\_a)  
 self.poke(pos\_a[0], pos\_a[1], self.peek(\*pos\_b))  
 self.poke(pos\_b[0], pos\_b[1], temp)  
  
  
def heur(puzzle, item\_total\_calc, total\_calc):  
 t = 0  
 for row in range(3):  
 for col in range(3):  
 val = puzzle.peek(row, col) - 1  
 target\_col = val % 3  
 target\_row = val / 3  
  
 *# account for 0 as blank* if target\_row < 0:  
 target\_row = 2  
  
 t += item\_total\_calc(row, target\_row, col, target\_col)  
  
 return total\_calc(t)  
  
  
def h\_manhattan(puzzle):  
 return heur(puzzle,  
 lambda r, tr, c, tc: abs(tr - r) + abs(tc - c),  
 lambda t: t)  
  
  
def h\_manhattan\_lsq(puzzle):  
 return heur(puzzle,  
 lambda r, tr, c, tc: (abs(tr - r) + abs(tc - c)) \*\* 2,  
 lambda t: math.sqrt(t))  
  
  
def h\_linear(puzzle):  
 return heur(puzzle,  
 lambda r, tr, c, tc: math.sqrt(math.sqrt((tr - r) \*\* 2 + (tc - c) \*\* 2)),  
 lambda t: t)  
  
  
def h\_linear\_lsq(puzzle):  
 return heur(puzzle,  
 lambda r, tr, c, tc: (tr - r) \*\* 2 + (tc - c) \*\* 2,  
 lambda t: math.sqrt(t))  
  
  
def h\_default(puzzle):  
 return 0  
  
  
def main():  
 p = EightPuzzle()  
 p.shuffle(20)  
 print(p)  
 path, count = p.solve(h\_manhattan)  
 path.reverse()  
 p.set("123405678")  
 print(p)  
 for i in path:  
 print(i)  
 print("distance is", count, "states")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 print("Iqrar Ijaz 01-131182-021")  
 main()

Output:

Graphical user interface, text, application

Description automatically generated

Graphical user interface, application

Description automatically generated