```python
# -*- coding: utf-8 -*-
import copy
import random
random.seed(1000)

max_step = 2000
class Node:
    def __init__(self, puzz, fvalue, gvalue, hvalue, parentnode):
        self.puzz = puzz
        self.gvalue = gvalue
        self.fvalue = fvalue
        self.hvalue = hvalue
        self.parentnode = parentnode


    def move(self, x1, y1, x2, y2, board):
        copyboard = copy.deepcopy(board)
        temp = copyboard[x1][y1]
        if x2 >= 0 and x2 < len(board) and y2 >= 0 and y2 < len(board):
            copyboard[x1][y1] = copyboard[x2][y2]
            copyboard[x2][y2] = temp
            return copyboard
        else:
            return -1


def search(x, puzzle):
    for i in range(len(puzzle)):
        for j in range(len(puzzle[i])):
            if puzzle[i][j] == x:
                return [i, j]
    print(f"error ,not exist {x}")
    return -1


def extend(node, goal,ls_open_node,heuristic_func):
    x, y = search("_", node.puzz)
    trylist = [[x - 1, y], [x + 1, y], [x, y + 1], [x, y - 1]]  # try possible movement with
    order  up down left right
    for i in trylist:
        resultboard = node.move(x, y, i[0], i[1], node.puzz)
        if resultboard != -1:
            newnode = Node(resultboard, 0, node.gvalue + 1, 0, node)
            newnode.hvalue = heuristic_func(newnode.puzz, goal)
            newnode.fvalue = newnode.hvalue + newnode.gvalue
            ls_open_node.append(newnode)

def heuristic_1(startboard, goalboard):
    """
    Counts the number of misplaced tiles
    """
    misplaced = 0
    for i in range(len(startboard)):
        for j in range(len(startboard[i])):
            if startboard[i][j] != "_" and startboard[i][j] != goalboard[i][j]:
                misplaced += 1
    return misplaced

def heuristic_2(startboard, goalboard):
    totaldistance = 0
    for i in range(0, len(startboard)):
        for j in range(0, len(startboard)):
            if startboard[i][j] != "_":
                totaldistance += abs(search(startboard[i][j], startboard)[0] -
                search(startboard[i][j], goalboard)[0])
```

```
                totaldistance += abs(search(startboard[i][j], startboard)[1] -
                search(startboard[i][j], goalboard)[1])
    return totaldistance

def heuristic_3(startboard, goalboard):
    totaldistance = 0
    for i in range(0, len(startboard)):
        for j in range(0, len(startboard)):
            if startboard[i][j] != "_":
                x1,y1 = search(startboard[i][j], startboard)
                x2,y2 = search(startboard[i][j], goalboard)
                totaldistance += pow(pow(x1-x2,2)+pow(y1-y2,2),0.5)
                #totaldistance += abs(search(startboard[i][j], startboard)[0] -
                search(startboard[i][j], goalboard)[0])
                #totaldistance += abs(search(startboard[i][j], startboard)[1] -
                search(startboard[i][j], goalboard)[1])
    return totaldistance

def findmin(openlist, closednodelist):
    minnode = Node(openlist[0].puzz, 800, 0, 0, None)
    for i in openlist:
        if i.fvalue < minnode.fvalue and i.puzz not in closednodelist:
            minnode = i
    return minnode


def print_list(list1):
    print("SUCCESS")
    print("")
    for i in list1:
        for k in i:
            line = ""
            for j in range(0, len(k)):
                if j == len(k) - 1:
                    line += str(k[j])
                else:
                    line = line + str(k[j]) + " "
            print(line)
        print("")


def findnode(node, list1):
    for i in list1:
        if node.puzz == i:
            return True
    return False


def path(node):
    pathlist = []
    pathlist.insert(0, node.puzz)
    while True:
        if node.parentnode == None or node == None:
            return pathlist
            break
        else:
            pathlist.insert(0, node.parentnode.puzz)
            node = node.parentnode

def generate_random_puzzle(puzzle_size = 3):
    """
    num:need generate puzzle number
    puzzle_size:puzzle size
    """
    tiles = []
```

```
    for i in range(1,puzzle_size*puzzle_size):
        tiles.append(str(i))
    tiles.append("_")
    random.shuffle(tiles)
    puzzle = []
    for i in range(puzzle_size):
        row = []
        for j in range(puzzle_size):
            row.append(tiles[puzzle_size*i+j])
        puzzle.append(row)
    return puzzle


def main(num = 1,puzzle_size = 3):
    if puzzle_size == 3:
        goal_tmp = [['_', '1', '2'], ['3', '4', '5'], ['6', '7', '8']]
        print(f"Goal {goal_tmp}")
        print("Number of puzzle:",num)
        print("{:<8}{:-^51}
        {:<10}{:<10}{:<10}{:<12}{:<12}{:<12}".format("Index","8-Puzzle","H1_step","H2_step","H3_s
        tep","H1_expend","H2_expend","H3_expend"))
    if puzzle_size == 4:
        goal_tmp = [['_', '1', '2', '3'], ['4', '5', '6', '7'], ['8', '9', '10', '11'], ['12',
        '13', '14', '15']]
        print(f"\nGoal {goal_tmp}")
        print("Number of puzzle:",num)
        print("{:<8}{:-^93}
        {:<10}{:<10}{:<10}{:<12}{:<12}{:<12}".format("Index","15-Puzzle","H1_step","H2_step","H3_
        step","H1_expend","H2_expend","H3_expend"))
    if puzzle_size == 5:
        goal_tmp = [['_', '1', '2', '3', '4'], ['5', '6', '7', '8', '9'], ['10', '11', '12',
        '13', '14'], ['15', '16', '17', '18', '19'], ['20', '21', '22', '23', '24']]
        print(f"\nGoal {goal_tmp}")
        print("Number of puzzle:",num)
        print("{:<8}{:-^149}
        {:<10}{:<10}{:<10}{:<12}{:<12}{:<12}".format("Index","24-Puzzle","H1_step","H2_step","H3_
        step","H1_expend","H2_expend","H3_expend"))

    functions = [heuristic_1,heuristic_2,heuristic_3]

    for i in range(num):
        start = generate_random_puzzle(puzzle_size)
        print("{:^5}   {}   ".format(i,start),end='')

        expend_list = []

        for func in functions:
            goal = copy.deepcopy(goal_tmp)
            ls_open_node = []  # not extendend node append this list as node
            closednodelist = []  # extended node append this list


            lensuccess = len(start[0])
            for i in start:
                if len(i) != lensuccess:
                    #print("FAILURE")
                    return
            firstnode = Node(start, 0, 0, -1, None)
            extend(firstnode, goal,ls_open_node,func)
            closednodelist.append(firstnode)
            while len(ls_open_node) > 0:
                newwillextendchild = findmin(ls_open_node, closednodelist)
                closednodelist.append(newwillextendchild.puzz)
                ls_open_node.remove(newwillextendchild)
                if len(closednodelist) >= max_step:
```

```
                    break
                if newwillextendchild.hvalue == 0:
                    break
                else:
                    extend(newwillextendchild, goal,ls_open_node,func)
            solutionlist = path(newwillextendchild)
            print("{:^7}   ".format(len(solutionlist)-1),end='')
            expend_list.append(len(closednodelist))
        for i in range(len(expend_list)):
            print("{:^9}   ".format(expend_list[i]),end='')
        print()
    print("{:-^150}\n".format("End"))


if __name__ == "__main__":
    main(100,3)
    main(100,4)
    main(100,5)

    # main(1,3)
    # main(1,4)
    # main(1,5)
```