25 Distinct Initial States of 9-Puzzle (empty slot is denoted by ‘X’):

1)
```
-----
| 1 |
-------------
| 4 | 6 | 2 |
-------------
| 5 | X | 3 |
-------------
| 7 | 8 | 9 |
-------------
```

2)
```
-----
| 1 |
-------------
| 2 | 6 | 8 |
-------------
| 4 | X | 3 |
-------------
| 7 | 9 | 5 |
-------------
```

3)
```
-----
| 1 |
-------------
| 3 | 4 | 6 |
-------------
| 2 | 5 | X |
-------------
| 7 | 8 | 9 |
-------------
```

4)
```
-----
| 1 |
-------------
| 4 | 3 | 6 |
-------------
| X | 2 | 9 |
-------------
| 5 | 7 | 8 |
-------------
```

5)
```
-----
| 1 |
-------------
| 4 | 2 | 5 |
-------------
| X | 6 | 3 |
-------------
| 7 | 8 | 9 |
-------------
```

6)
```
-----
| 1 |
-------------
| 4 | X | 3 |
-------------
| 7 | 2 | 6 |
-------------
| 8 | 5 | 9 |
-------------
```

7)
```
-----
| 1 |
-------------
| 4 | 6 | 2 |
-------------
| 7 | X | 3 |
-------------
| 8 | 5 | 9 |
-------------
```

8)
```
-----
| 1 |
-------------
| 5 | 7 | 3 |
-------------
| 2 | 4 | X |
-------------
| 8 | 9 | 6 |
-------------
```

9)
```
-----
| 1 |
-------------
| 2 | 6 | 7 |
-------------
| 4 | 3 | 9 |
-------------
| 8 | X | 5 |
-------------
```

10)
```
-----
| 1 |
-------------
| 2 | X | 5 |
-------------
| 4 | 6 | 3 |
-------------
| 7 | 8 | 9 |
-------------
```

11)
```
-----
| 1 |
-------------
| 3 | 5 | 6 |
-------------
| 2 | 9 | X |
-------------
| 4 | 7 | 8 |
-------------
```

12)
```
-----
| 1 |
-------------
| 2 | 5 | 3 |
-------------
| 7 | X | 6 |
-------------
| 8 | 4 | 9 |
-------------
```

13)
```
-----
| 1 |
-------------
| 4 | 2 | 3 |
-------------
| 7 | X | 9 |
-------------
| 8 | 6 | 5 |
-------------
```

14)
```
-----
| 1 |
-------------
| 4 | 6 | 2 |
-------------
| 5 | 8 | 3 |
-------------
| 7 | X | 9 |
-------------
```

15)
```
-----
| X |
-------------
| 1 | 2 | 3 |
-------------
| 4 | 5 | 6 |
-------------
| 7 | 8 | 9 |
-------------
```

16)
```
-----
| 1 |
-------------
| 2 | 5 | 3 |
-------------
| 4 | 8 | X |
-------------
| 7 | 9 | 6 |
-------------
```

17)
```
-----
| 1 |
-------------
| X | 2 | 3 |
-------------
| 4 | 5 | 6 |
-------------
| 7 | 8 | 9 |
-------------
```

18)
```
-----
| 1 |
-------------
| 2 | 5 | 3 |
-------------
| 4 | 9 | 8 |
-------------
| 7 | X | 6 |
-------------
```

19)
```
-----
| 1 |
-------------
| 2 | X | 6 |
-------------
| 5 | 3 | 9 |
-------------
| 4 | 7 | 8 |
-------------
```

20)
```
-----
| 1 |
-------------
| X | 2 | 5 |
-------------
| 3 | 4 | 6 |
-------------
| 7 | 8 | 9 |
-------------
```

21)
```
-----
| 1 |
-------------
| 4 | 2 | 3 |
-------------
| 5 | 9 | 8 |
-------------
| 7 | X | 6 |
-------------
```

22)
```
-----
| 1 |
-------------
| 5 | 3 | 6 |
-------------
| 4 | 7 | 9 |
-------------
| 2 | X | 8 |
-------------
```

23)
```
-----
| 1 |
-------------
| 2 | 5 | 3 |
-------------
| 8 | 7 | 6 |
-------------
| 4 | X | 9 |
-------------
```

24)
```
-----
| 1 |
-------------
| 5 | 6 | 4 |
-------------
| 7 | X | 2 |
-------------
| 8 | 9 | 3 |
-------------
```

25)
```
-----
| 1 |
-------------
| 4 | 2 | 3 |
-------------
| 8 | 7 | 6 |
-------------
| X | 5 | 9 |
-------------
```

Source code:

---

solution.py

```
from state import print_solution_path
from beam_search import beam_search
from puzzle import generate
from write_to_file import write_to_csv, write_to_txt

'''
    CS461 - Artificial Intelligence Homework 2

    Group members:
        * Fuad Aghazada
        * Can Özgürel
        * Çağatay Sel
        * Utku Mert Topçuoğlu
        * Kaan Kıranbay

    As heuristic function
        h2 (sum of Manhattan distances of the tiles from their goal positions)
        has been used

    @authors: fuadaghazada, canozgurel
    @date: 7/3/2019
'''

'''
    Generating N distinct puzzles
'''
def generate_n_puzzles(n):
    distinct_puzzles = []

    while len(distinct_puzzles) != n:
        puzzle = generate()

        if puzzle not in distinct_puzzles:
            distinct_puzzles.append(puzzle)

    return distinct_puzzles


###### GENERATING 25 Distinct Puzzles #######
puzzles = generate_n_puzzles(25)

index = 1
path = None

# X and Y values for the puzzle solved with beam width 2 and 3
data = []

for puzzle in puzzles:
    path, num_moves_w2 = beam_search(puzzle, 2)
    path, num_moves_w3 = beam_search(puzzle, 3)

    data.append([index, num_moves_w2, num_moves_w3])
    index += 1
```

```
# Write puzzles (initial states) to txt
write_to_txt(puzzles)

# Writing result into csv file
write_to_csv(data)

# Print the trace for the last puzzle
print("\n\n----Solution trace for last puzzle!----\n\n")
if path:
    print_solution_path(path)
```

beam_search.py

```
import math

from state import generate_next_states
from puzzle import GOAL, find_index

'''
    Searching for the solution

    @param:
        - given puzzle
        - beam width

    @return:
        - solution_path
        - number of moves
'''
def beam_search(puzzle, w):

    # Queue with one element
    queue = list()
    queue.append([puzzle])

    while len(queue) != 0:
        first = queue.pop(0)

        paths = []

        # Next states of the last node of the popped path
        next_states = generate_next_states(first[-1])

        for state in next_states:
            path = tuple(first)

            # Rejecting paths with loops
            if state not in path:
                path = list(path)
                path.append(state)
                paths.append({'path': path, 'h_value': manhattan_distance(path[-1])})

        # Sorting according to heuristic value of path
        paths = sorted(paths, key = lambda k: k['h_value'])

        # Adding new pathes to the 'front' of queue
        for path in paths[:w]:
            queue.append(path['path'])
```

```
        # Check if goal is found!
        if len(queue) != 0 and GOAL in queue[0]:
            num_moves = len(queue[0]) - 1

            print('--------------------------')
            print("Solved with beam width: ", w)
            print("Number of moves: ", num_moves)
            print('--------------------------\n')

            return queue[0], num_moves;

        elif len(queue) == 0:
            print("No path is found\n")

    return None, None;


'''
    Finding manhattan_distance for a given puzzle
'''
def manhattan_distance(state):
    distance = 0

    for i in range(0, len(state)):
        for j in range(0, len(state[i])):
            value = state[i][j]

            if value is 'X':
                continue

            m, n = find_index(GOAL, value)

            distance += math.fabs(i - m)
            distance += math.fabs(j - n)

    return distance
```

puzzle.py

```
import random
import copy

# Goal puzzle
GOAL = [['X'],
        ['1', '2', '3'],
        ['4', '5', '6'],
        ['7', '8', '9']]

'''
    Generating random puzzle
'''
def generate():
    no_steps = random.randint(25, 30)

    puzzle = copy.deepcopy(GOAL)

    # Moving empty slot random number of times
```

```
    for i in range(no_steps):
        puzzle = move_random(puzzle)

    return puzzle


'''
    Finding any tile indices from the puzzle
'''
def find_index(puzzle, value):
    for i in range(len(puzzle)):
        for j in range(len(puzzle[i])):
            if puzzle[i][j] == value:
                return (i, j)
    return None


'''
    Finding neighbors of a tile with the given index
    in the puzzle
'''
def find_possible_moves(puzzle):
    # Location of empty slot
    (i, j) = find_index(puzzle, 'X')

    # Possible moves
    moves = []

    # Up
    if i > 1 or (i == 1  and j == 0):
        moves.append('up')

    # Down
    if i < len(puzzle) - 1:
        moves.append('down')

    # Right
    if i > 0 and j < len(puzzle[i]) - 1:
        moves.append('right')

    # Left
    if i > 0 and j > 0:
        moves.append('left')

    return moves


'''
    Randomly move the empty tile to an available spot
'''
def move_random(puzzle):
    # Possible moves
    moves = find_possible_moves(puzzle)

    # Choice index
    choice = moves[random.randint(0, len(moves) - 1)]

    # Moving the tile according to random choice
    return move(puzzle, choice)
```

```
'''
    Moving a tile in the given direction
'''
def move(puzzle, dir):
    # Location of Empty slot
    (x_i, x_j) = find_index(puzzle, 'X')

    puzzle_cp = copy.deepcopy(puzzle)

    # Apply move
    if dir is 'up':
        puzzle_cp[x_i - 1][x_j], puzzle_cp[x_i][x_j] = puzzle_cp[x_i][x_j], puzzle_cp[x_i - 1][x_j
    elif dir is 'down':
        puzzle_cp[x_i + 1][x_j], puzzle_cp[x_i][x_j] = puzzle_cp[x_i][x_j], puzzle_cp[x_i + 1][x_j
    elif dir is 'right':
        puzzle_cp[x_i][x_j + 1], puzzle_cp[x_i][x_j] = puzzle_cp[x_i][x_j], puzzle_cp[x_i][x_j + 1
    elif dir is 'left':
        puzzle_cp[x_i][x_j - 1], puzzle_cp[x_i][x_j] = puzzle_cp[x_i][x_j], puzzle_cp[x_i][x_j - 1

    return puzzle_cp
```

state.py

```
from puzzle import find_index, find_possible_moves, move, GOAL

'''
    Generating next states for the given state of puzzle

    Note: states are sorted according to their distance (Manhattan) to goal
'''
def generate_next_states(puzzle):

    next_states = []
    moves = find_possible_moves(puzzle)

    for m in moves:
        state = move(puzzle, m)
        next_states.append(state)

    return next_states


'''
    Get the move type between 2 states: from state1 to state2
'''
def move_statement(state1, state2):
    (i1, j1) = find_index(state1, 'X')
    (i2, j2) = find_index(state2, 'X')

    dh = j2 - j1
    dv = i2 - i1

    moved_tile = state1[i2][j2]
    statement = str(moved_tile) + " moved "

    if dh == 1:
        statement += 'left'
    elif dh == -1:
```

```
        statement += 'right'
    elif dv == 1:
        statement += 'up'
    elif dv == -1:
        statement += 'down'
    else:
        statement = ""

    return statement + "\n"


'''
    Printing the puzzle state in a pretty format
'''
def print_state(state):
    print('-----')
    for i in state:
        for j in i:
            if j is i[0]:
                print('| ', end='')
            print(j + ' | ', end='')
        print('\n-------------')
    print('\n')


'''
    Printing the solution path
'''
def print_solution_path(path):
    cur_state = None
    for state in path:
        if cur_state:
            print(move_statement(cur_state, state))
        print_state(state)
        print("-----------------")
        cur_state = state
```

write_to_file.py

```
import csv

'''
    Writing the given data in rows to a csv file
'''
def write_to_csv(data):
    with open('../data/result.csv', 'w') as writeFile:
        writer = csv.writer(writeFile)
        writer.writerow(['Puzzle #', '# of moves (beam width 2)', '# of moves (beam
width 3)'])
        for row in data:
            writer.writerow(row)

'''
    Writing the puzzles to a text file in a 'pretty format'
'''
def write_to_txt(puzzles):
    with open('../data/puzzles.txt', 'w') as writeFile:
        index = 1
        for puzzle in puzzles:
            writeFile.write(str(index) + ")\n")
```

```
        writeFile.write('-----\n')
        for i in puzzle:
            for j in i:
                if j is i[0]:
                    writeFile.write('| ')
                writeFile.write(j + ' | ')
            writeFile.write('\n-------------\n')
        writeFile.write('\n')
        index += 1
```

Result Graph: