```python
#Manhattan approach
import heapq
from termcolor import colored

class PuzzleState:
    def __init__(self, board, parent, move, depth):
        self.board = board
        self.parent = parent
        self.move = move
        self.depth = depth
        self.heuristic = heuristic(board)
        self.cost = self.depth + self.heuristic

    def __lt__(self, other):
        return self.cost < other.cost

def print_board(board):
    print("+---+---+---+")
    for row in range(0, 9, 3):
        row_visual = "|"
        for tile in board[row:row + 3]:
            if tile == 0:
                row_visual += f" {colored(' ', 'cyan')} |"
            else:
                row_visual += f" {colored(str(tile), 'yellow')} |"
        print(row_visual)
        print("+---+---+---+")

goal_state = [1,2,3,8,0,4,7,6,5]

moves = {
    'U': -3,
    'D': 3,
    'L': -1,
    'R': 1
}

def heuristic(board):
    distance = 0
    for i in range(9):
        if board[i] != 0:
            x1, y1 = divmod(i, 3)
            x2, y2 = divmod(board[i] - 1, 3)
            distance += abs(x1 - x2) + abs(y1 - y2)
    return distance
```

```python
def move_tile(board, move, blank_pos):
    new_board = board[:]
    new_blank_pos = blank_pos + moves[move]
    new_board[blank_pos], new_board[new_blank_pos] = new_board[new_blank_pos], new_board[blank_pos]
    return new_board

def a_star(start_state):
    open_list = []
    closed_list = set()
    heapq.heappush(open_list, PuzzleState(start_state, None, None, 0))

    while open_list:
        current_state = heapq.heappop(open_list)

        if current_state.board == goal_state:
            return current_state

        closed_list.add(tuple(current_state.board))

        blank_pos = current_state.board.index(0)

        for move in moves:
            if move == 'U' and blank_pos < 3:
                continue
            if move == 'D' and blank_pos > 5:
                continue
            if move == 'L' and blank_pos % 3 == 0:
                continue
            if move == 'R' and blank_pos % 3 == 2:
                continue

            new_board = move_tile(current_state.board, move, blank_pos)

            if tuple(new_board) in closed_list:
                continue

            new_state = PuzzleState(new_board, current_state, move, current_state.depth + 1)
            heapq.heappush(open_list, new_state)

    return None
```

```python
def print_solution(solution):
    path = []
    current = solution
    while current:
        path.append(current)
        current = current.parent
    path.reverse()

    for step in path:
        print(f"Move: {step.move}")
        print_board(step.board)

    total_cost = solution.depth
    print(colored(f"Total cost to reach the goal node (g(n)): {total_cost}", "green"))

initial_state = [2,8,3,1,6,4,7,0,5]

solution = a_star(initial_state)

if solution:
    print(colored("Solution found:", "green"))
    print_solution(solution)
else:
    print(colored("No solution exists.", "red"))
print("Goal reached")
```

```
Solution found:
Move: None
+---+---+---+
| 2 | 8 | 3 |
+---+---+---+
| 1 | 6 | 4 |
+---+---+---+
| 7 |   | 5 |
+---+---+---+
Move: U
+---+---+---+
| 2 | 8 | 3 |
+---+---+---+
| 1 |   | 4 |
+---+---+---+
| 7 | 6 | 5 |
+---+---+---+
Move: U
+---+---+---+
| 2 |   | 3 |
+---+---+---+
| 1 | 8 | 4 |
+---+---+---+
| 7 | 6 | 5 |
+---+---+---+
Move: L
+---+---+---+
|   | 2 | 3 |
+---+---+---+
| 1 | 8 | 4 |
+---+---+---+
| 7 | 6 | 5 |
+---+---+---+
Move: D
+---+---+---+
| 1 | 2 | 3 |
+---+---+---+
|   | 8 | 4 |
+---+---+---+
| 7 | 6 | 5 |
+---+---+---+
Move: R
+---+---+---+
| 1 | 2 | 3 |
+---+---+---+
| 8 |   | 4 |
+---+---+---+
| 7 | 6 | 5 |
+---+---+---+
Total cost to reach the goal node (g(n)): 5
Goal reached
```

```
Enter the No. of Rows: 1
Enter the No. of Columns: 2
Enter clean status for each cell (1 - dirty, 0 - clean):
Enter value for cell (1, 1): 0
Enter value for cell (1, 2): 1

The Floor matrix is as below:
 >0<   1

The Floor matrix is as below:
  0   >1<

The Floor matrix is as below:
  0   >0<

Total cost is: 1
Goal achieved
```

```python
#Misplaced tiles approach
import heapq
from termcolor import colored

class PuzzleState:
    def __init__(self, board, parent, move, depth):
        self.board = board
        self.parent = parent
        self.move = move
        self.depth = depth
        self.heuristic = misplaced_tiles(board)
        self.cost = self.depth + self.heuristic

    def __lt__(self, other):
        return self.cost < other.cost

def print_board(board):
    print("+---+---+---+")
    for row in range(0, 9, 3):
        row_visual = "|"
        for tile in board[row:row + 3]:
            if tile == 0:
                row_visual += f" {colored(' ', 'cyan')} |"
            else:
                row_visual += f" {colored(str(tile), 'yellow')} |"
        print(row_visual)
        print("+---+---+---+")

goal_state = [1,2,3,8,0,4,7,6,5]

moves = {
    'U': -3,
    'D': 3,
    'L': -1,
    'R': 1
}

def misplaced_tiles(board):
    return sum(1 for i in range(9) if board[i] != goal_state[i] and board[i] != 0)

def move_tile(board, move, blank_pos):
    new_board = board[:]
    new_blank_pos = blank_pos + moves[move]
    new_board[blank_pos], new_board[new_blank_pos] = new_board[new_blank_pos], new_board[blank_pos]
    return new_board
```

```python
def a_star(start_state):
    open_list = []
    closed_list = set()
    heapq.heappush(open_list, PuzzleState(start_state, None, None, 0))

    while open_list:
        current_state = heapq.heappop(open_list)

        if current_state.board == goal_state:
            return current_state

        closed_list.add(tuple(current_state.board))

        blank_pos = current_state.board.index(0)

        for move in moves:
            if move == 'U' and blank_pos < 3:
                continue
            if move == 'D' and blank_pos > 5:
                continue
            if move == 'L' and blank_pos % 3 == 0:
                continue
            if move == 'R' and blank_pos % 3 == 2:
                continue

            new_board = move_tile(current_state.board, move, blank_pos)

            if tuple(new_board) in closed_list:
                continue

            new_state = PuzzleState(new_board, current_state, move, current_state.depth + 1)
            heapq.heappush(open_list, new_state)

    return None

def print_solution(solution):
    path = []
    current = solution
    while current:
        path.append(current)
        current = current.parent
    path.reverse()

    for step in path:
        print(f"Move: {step.move}")
        print_board(step.board)

    total_cost = solution.depth
    print(colored(f"Total cost to reach the goal node (g(n)): {total_cost}", "green"))

initial_state = [2,8,3,1,6,4,7,0,5]
```

```python
solution = a_star(initial_state)

if solution:
    print(colored("Solution found:", "green"))
    print_solution(solution)
else:
    print(colored("No solution exists.", "red"))
```

```
Solution found:
Move: None
+---+---+---+
| 2 | 8 | 3 |
+---+---+---+
| 1 | 6 | 4 |
+---+---+---+
| 7 |   | 5 |
+---+---+---+
Move: U
+---+---+---+
| 2 | 8 | 3 |
+---+---+---+
| 1 |   | 4 |
+---+---+---+
| 7 | 6 | 5 |
+---+---+---+
Move: U
+---+---+---+
| 2 |   | 3 |
+---+---+---+
| 1 | 8 | 4 |
+---+---+---+
| 7 | 6 | 5 |
+---+---+---+
Move: L
+---+---+---+
|   | 2 | 3 |
+---+---+---+
| 1 | 8 | 4 |
+---+---+---+
| 7 | 6 | 5 |
+---+---+---+
Move: D
+---+---+---+
| 1 | 2 | 3 |
+---+---+---+
|   | 8 | 4 |
+---+---+---+
| 7 | 6 | 5 |
+---+---+---+
Move: R
+---+---+---+
| 1 | 2 | 3 |
+---+---+---+
| 8 |   | 4 |
+---+---+---+
| 7 | 6 | 5 |
+---+---+---+
Total cost to reach the goal node (g(n)): 5
```