Problem Description:

As you all know about 8-puzzle game discussed in the AI class.
You must write a function called *Create (Initial state, Level number)* to generate all the states of 8-puzzle game starting from the given initial state up to a particular level. The initial state and level number are the input to the *Create* function. Use the suitable data structure to represent the state of the 8-puzzle game and use the actions (Up, Down, Left, and Right) to generate the new states. Print all the states which are generated by the *Create* function.

CODE:

```
from collections import deque

# Define the goal state
goal_state = [
    [1, 2, 3],
    [4, 0, 5],
    [6, 7, 8]
]

# Define the possible moves (Up, Down, Left, Right)
moves = [(0, -1), (0, 1), (-1, 0), (1, 0)]  # (row_change, col_change)

def create(initial_state, level):
    def is_valid_move(x, y):
        return 0 <= x < 3 and 0 <= y < 3

    def apply_move(state, x, y, new_x, new_y):
        new_state = [list(row) for row in state]
        new_state[x][y], new_state[new_x][new_y] = new_state[new_x][new_y], new_state[x][y]
        return new_state

    def print_state(state):
        for row in state:
            print(" ".join(map(str, row)))
        print()

    initial_state = [list(row) for row in initial_state]
```

```
    visited = set()
    queue = deque([(initial_state, 0)])

    while queue:
        current_state, current_level = queue.popleft()

        if current_level > level:
            break

        if current_state == goal_state:
            print("Goal state reached at level", current_level)
            print_state(current_state)
            break

        print("Level", current_level)
        print_state(current_state)
        visited.add(tuple(tuple(row) for row in current_state))

        for dx, dy in moves:
            x, y = None, None
            for i in range(3):
                for j in range(3):
                    if current_state[i][j] == 0:
                        x, y = i, j
                        break

            new_x, new_y = x + dx, y + dy

            if is_valid_move(new_x, new_y):
                new_state = apply_move(current_state, x, y, new_x, new_y)
                new_state_tuple = tuple(tuple(row) for row in new_state)
                if new_state_tuple not in visited:
                    queue.append((new_state, current_level + 1))

# Example usage:
initial_state = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 0]
]
```

level = 20 # You can adjust the level as needed
create(initial_state, level)

OUTPUT:

```
Level 0
1 2 3
4 5 6
7 8 0

Level 1
1 2 3
4 5 6
7 0 8

Level 1
1 2 3
4 5 0
7 8 6

Level 2
1 2 3
4 5 6
0 7 8
```

```
Level 2
1 2 3
4 0 6
7 5 8

Level 2
1 2 3
4 0 5
7 8 6

Level 2
1 2 0
4 5 3
7 8 6

Level 3
1 2 3
0 5 6
4 7 8
```

```
Level 3
1 2 3
0 4 6
7 5 8

Level 3
1 2 3
4 6 0
7 5 8

Level 3
1 0 3
4 2 6
7 5 8

Level 3
1 2 3
0 4 5
7 8 6
```

```
Level 3
1 0 3
4 2 5
7 8 6

Level 3
1 2 3
4 8 5
7 0 6

Level 3
1 0 2
4 5 3
7 8 6

Level 4
1 2 3
5 0 6
4 7 8
```

```
Level 4
0 2 3
1 5 6
4 7 8

Level 4
0 2 3
1 4 6
7 5 8

Level 4
1 2 3
7 4 6
0 5 8

Level 4
1 2 0
4 6 3
7 5 8
```

```
Level 4
1 2 3
4 6 8
7 5 0

Level 4
0 1 3
4 2 6
7 5 8

Level 4
1 3 0
4 2 6
7 5 8

Level 4
0 2 3
1 4 5
7 8 6
```

```
Level 4
1 2 3
7 4 5
0 8 6

Level 4
0 1 3
4 2 5
7 8 6

Level 4
1 3 0
4 2 5
7 8 6

Level 4
1 2 3
4 8 5
0 7 6
```