Implementation of Iterative Deepening Search Algorithm to solve 8-Puzzle Problem

```
class PuzzleState:
    def __init__(self, board, empty_tile_pos, moves=0, prev=None, direction=None):
        self.board = board
        self.empty_tile_pos = empty_tile_pos
        self.moves = moves
        self.prev = prev
        self.direction = direction
        self.size = 3  # 3x3 board

    def is_goal(self):
        return self.board == [1, 2, 3, 4, 5, 6, 7, 8, 0]

    def get_possible_moves(self):
        moves = []
        row, col = divmod(self.empty_tile_pos, self.size)
        directions = [(-1, 0, 'Up'), (1, 0, 'Down'), (0, -1, 'Left'), (0, 1, 'Right')]  # Up, Down, Left, Right

        for dr, dc, dir_name in directions:
            new_row, new_col = row + dr, col + dc
            if 0 <= new_row < self.size and 0 <= new_col < self.size:
                new_empty_tile_pos = new_row * self.size + new_col
                moves.append((new_empty_tile_pos, dir_name))
```

```
        return moves

    def move(self, new_empty_tile_pos, direction):
        new_board = self.board[:]
        new_board[self.empty_tile_pos], new_board[new_empty_tile_pos] = new_board[new_empty_tile_pos], new_board[self.empty_tile_pos]
        return PuzzleState(new_board, new_empty_tile_pos, self.moves + 1, self, direction)

    def __repr__(self):
        return '\n'.join([' '.join([str(self.board[i * self.size + j]) for j in range(self.size)]) for i in range(self.size)]) + f'\nMoves: {self.moves}'

def depth_limited_search(state, limit):
    if state.is_goal():
        return state
    if state.moves >= limit:
        return None

    for new_empty_tile_pos, direction in state.get_possible_moves():
        new_state = state.move(new_empty_tile_pos, direction)
        result = depth_limited_search(new_state, limit)
        if result:
            return result
```

```
    return None

def iterative_deepening_search(initial_state, limit):
    for depth in range(0, limit + 1, 2):  # Increase depth by 2 for even limits
        print(f"Searching at depth: {depth}")
        states_at_level = []
        result = depth_limited_search_with_states(initial_state, depth,
states_at_level)
        print_states(states_at_level, depth)
        if result:
            return result
    return None

def depth_limited_search_with_states(state, limit, states_at_level):
    if state.is_goal():
        return state
    if state.moves >= limit:
        return None

    # Store the state and direction
    states_at_level.append((state, state.direction))

    for new_empty_tile_pos, direction in state.get_possible_moves():
        new_state = state.move(new_empty_tile_pos, direction)
        result = depth_limited_search_with_states(new_state, limit,
states_at_level)
```

```
        if result:
            return result

    return None

def print_states(states_at_level, depth):
    if not states_at_level:
        print(f"No states found at depth {depth}.")
        return

    print(f"States at depth {depth}:")
    for state, direction in states_at_level:
        print(f"Direction: {direction}, State:\n{state}\n")

def main():
    # Get user input for the initial state of the puzzle
    print("Enter the initial state of the 8-puzzle (use 0 for the empty tile):")
    initial_board = list(map(int, input().strip().split()))

    # Validate input
    if len(initial_board) != 9 or set(initial_board) != set(range(9)):
        print("Invalid input! Please enter 9 unique numbers from 0 to 8.")
        return

    empty_tile_pos = initial_board.index(0)
```

```
    initial_state = PuzzleState(initial_board, empty_tile_pos)

    # Get user input for limit
    limit = int(input("Enter the maximum depth limit (even number): "))
    if limit % 2 != 0:
        print("Limit must be an even number!")
        return

    # Run IDS
    solution = iterative_deepening_search(initial_state, limit)

    # Print the solution path
    if solution:
        path = []
        while solution:
            path.append(solution)
            solution = solution.prev
        for step in reversed(path):
            print(step)
    else:
        print("No solution found within the specified limit.")

if __name__ == "__main__":
    main()
```

Output:

Enter the initial state of the 8-puzzle (use 0 for the empty tile):

1 4 3 7 0 6 5 8 2

Enter the maximum depth limit (even number): 2

Searching at depth: 0

No states found at depth 0.

Searching at depth: 2

States at depth 2:

Direction: None, State:

1 4 3

7 0 6

5 8 2

Moves: 0

Direction: Up, State:

1 0 3

7 4 6

5 8 2

Moves: 1

Direction: Down, State:

1 4 3

7 8 6

5 0 2

Moves: 1

Direction: Left, State:

1 4 3

0 7 6

5 8 2

Moves: 1

Direction: Right, State:

1 4 3

7 6 0

5 8 2

Moves: 1

No solution found within the specified limit.