Artificial Intelligence HW2

By: Franklin Neves Filho
PID: 6208239

Problem 1 - *Number of Islands*

Pseudocode:

```
function bfs(grid, i, j):
    Create queue and add current coordinates
    while queue is not empty:
        current cell = queue.pop()
        If current cell is out of bounds or is water:
            continue to next in queue
        set the current cell to 0
        add to queue all 4 possible moves


function dfs(grid, i, j):
    base case:
        if coordinates are out of bounds or water return nothing
    set coordinate to 0
    call dfs for all 4 moves

function count_islands(grid=None, method='dfs'):
    island_map is defined globally and is copied if no map provided.

    if method is not dfs or bfs return error
    set initial count to 0
    for column in grid:
        for row in grid:
            if current coordinates is 1:
                count += 1
                if method is dfs:
                    dfs(grid, column, row)
                if method is bfs:
                    bfs(grid, column, row)
    return count
```

Time complexity:

O(m *n) where m is the number of rows and n is the number of columns in the grid.

Space complexity:

O(m *n) where m is the number of rows and n is the number of columns in the grid.

3. As a human I do count 17 islands, and the code returns 17 as well. The dfs and bfs function are used to find all the coordinates within the island and convert it into 0 or water. This way when iterating through the grid the function only counts the first occurrence of the island.

Problem 2 - *8-Puzzle Games*

Pseudocode:

```
Initialize target grid _target as:
   _target =     [[1, 2, 3],
                   [4, 5, 6],
                   [7, 8, 0]]

Function _get_distance(grid):
        Initialize distance as 0
        For each row i in the range 0 to 2:
                For each column j in the range 0 to 2:
                        If grid[i][j] is 0:
                                Skip this iteration
        Calculate the target position (x, y) of the current tile: **x, y = divmod(grid[i][j] - 1, 3)**
        Add to the distance: **distance += |x - i| + |y - j|** (This is the Manhattan distance)
        **Return distance**

Function sliding_puzzle(grid):
        Initialize an empty PriorityQueue pq
        Initialize an empty set seen (to keep track of visited grids)

        If grid is equal to _target:
                **Return 0** (No moves needed)
        Add the initial grid with priority 0 to pq:  **pq.put((0, grid))**

        While pq is not empty:
                Pop the top element (distance, current_grid) from pq
                Add the current_grid to seen set: seen.add(current_grid)

                If current_grid equals _target:
                        Return distance (Solution found)

                Find the position (i, j) of the empty tile (0) in current_grid:
                For i from 0 to 2:
                        For j from 0 to 2:
                                If current_grid[i][j] == 0:
                                        Break both loops (Found empty tile)

                For each possible move (x, y) in directions (right, down, left, up): **(0, 1), (1, 0), (-1, 0), (0, -1)**
                        If the new position (i + x, j + y) is within grid bounds:
                        Create a copy of current_grid as new_grid
                        Swap the empty tile (0) with the neighboring tile at (i + x, j + y) in new_grid

                        If new_grid has not been seen (not in seen):
                                Add new_grid to pq with priority (distance + 1):
                                pq.put((distance + 1, new_grid))

        If no solution is found:          Return None
```

Time Complexity: O(n $\log_n$)

Space Complexity: O(n)

n represents the number of unique states, n being 181,440

I specifically chose the Manhattan distance for the heuristic as it directly takes into account the up, down, left and right movements of the puzzle. It also ignores the diagonal which in this problem is needed to be ignored since we cannot move the pieces diagonally