## Intuition

The problem is asking us to find the minimum number of moves required to reach the end of a board game represented by a 2D grid (snakes and ladders game). The key observation here is that the board can be represented by a 1D array, where each element in the array corresponds to a cell in the 2D grid. This allows us to use a breadth-first search (BFS) algorithm to traverse the board and find the shortest path to the end.

The BFS algorithm works by starting from the first cell (0th index in the 1D array) and exploring all its neighboring cells that can be reached in one move (in this case, the next 6 cells in the 1D array). For each of these neighboring cells, we check if it has a snake or ladder and update the next cell accordingly. We also keep track of the distance (number of moves) to each cell and use this information to determine the minimum number of moves required to reach the end.

The algorithm continues this process until the end cell (last index in the 1D array) is reached, at which point the algorithm returns the minimum number of moves required to reach the end. If the end is not reached, the algorithm returns -1 indicating that it is impossible to reach the end.

# Approach

The approach used in this solution is Breadth First Search (BFS). It first flattens the board into a 1D array and then uses a queue to keep track of the current position and the distance from the start. The algorithm starts at the first position (0) and uses a loop to check all possible moves (1-6) from the current position. If the next position has not been visited yet, it updates the distance and pushes it into the queue. If the next position is a snake or ladder, it updates the position to the end of the snake or ladder. The algorithm continues this process until it reaches the end of the board or the queue is empty. If the end of the board is reached, it returns the distance, otherwise, it returns -1.

**Complexity**

- Time complexity:
  O(n2)O(n^2)$O(n2)$ :
  where n is the size of the board. This is because the algorithm must visit each cell in the board at most once, and the number of cells in the board is n^2.
- Space complexity:
  O(n2)O(n^2)$O(n2)$ :
  we use an array of size n^2 to store the flattened board and another array of size n^2 to store the distance from each cell to the starting cell. Additionally, we use a queue to keep track of the cells to be visited, which also takes up O(n^2) space in the worst case scenario.