Mine Sweeper - 2D Array with OOP / Recursion

Mine sweeper is a popular mini-game. In this exercise, you are required to use 2d-array to implement an OOP solution for Mine Sweeper.

https://minesweeper.online/

Create a class `MineSweeper` according to the following specifications:

Attributes	**Specifications**
`- size: int`	`size` of the board. If `size` is `3`, then the board will be a `3x3` matrix.
`- no_of_mines`	The number of mines in the board.
`- board: list`	The 2d `list` to store the board information.
`- gboard: list`	The 2d `list` to store the player guess information. It should have the same dimension as the board, but init with `None` values.
`- gstatus: bool`	A `boolean` variable saving game status: - `True` means game is still ongoing - `False` means game is finished (either win or lose)
Methods	**Specifications**
`__init__(self, size, no_of_mines)`	Init with the `size`, and set the `board` and `gboard` to a `n` by `n` `list` with `None` values, and set `no_of_mines` mines.
`reset(self)`	Reset the board with `None` values. Set `gstatus` to `True`.

`display(self, guess_mode=True)`	Display the 2d list with the following format:
	- if value is `None`, display a minus sign;
	- if value is 0, display an empty cell;
	- otherwise, display the value stored in the cell.
	
	- If `guess_mode` is `True` (by default), display the `gboard`;
	- If it is `False`, display the `board`.
	`+---+---+---+---+---+---+`
	`| 1 | 2 | 2 | 1 | 1 | 1 |`
	`+---+---+---+---+---+---+`
	`| 1 | @ | @ | 2 | 2 | @ |`
	`+---+---+---+---+---+---+`
	`| 1 | 3 | 3 | 3 | @ | 3 |`
	`+---+---+---+---+---+---+`
	`|  | 1 | @ | 2 | 3 | @ |`
	`+---+---+---+---+---+---+`
	`|  | 1 | 2 | 2 | 3 | @ |`
	`+---+---+---+---+---+---+`
	`|  |  | 1 | @ | 2 | 1 |`
	`+---+---+---+---+---+---+`

`read_game(self, filename)`	Read the game from a file. - if the char is a `@` symbol, read it as a mine, and increase the number of mines by 1. - if the char is a space, add a `None` value to the cell.

`save_game(self, filename)`	Write the game to a file. - for mines, save it as `@` symbol - for all other values, save as an empty space.
`generate_mines(self)`	Randomly generate mines across the board based on the `no_of_mines`.
`generate_numbers(self)`	Count and generate the numbers inside the board, the number represents the number of mines found within the 3x3 grid of the cell.
`check_win(self)`	Check if the player has won. - If all cells that are not mines have been revealed, return `True`; - Otherwise, return `False`.
`guess(self, row, col, action)`	Implement the following actions by the players at the position `(row, col)`: - `"f"` for flag: player place a flag at position indicating that he is guessing this position as a mine, Set the cell value to `"⚑"`. - `"u"` for unflag: remove the flag status. Set the cell to `None`. - `"c"` for click: - if it is a mine, print game over message and display the actual board. - if the cell is `0`, reveal all cells with number values surrounding it, this action is continue until all the connected areas have been revealed. - if the cell is not `0`, reveal the number. - check if the game has been won. - `"d"` for double click: - if the number of flagged cells surrounding the position equals to the digit of the cell, perform click action to all unflagged surrounding cells. If the game is not finished, display the guess board for the player to see the current progress.

- Create your own test cases when necessary to test out your code.
- Create an interactive text-based user interface for users to play the game, with the following features:
    - Start new game
    - User Actions
    - Save the game
    - Load the game