PROJECT 2 – Sher-Locked!

Important Notes:

1. If you do not have time to finish the project during the practical sessions, **you must finish it later**.
2. **Must be used:**
   a) **The same name we propose for the functions and procedures**
   b) **The same messages**

Otherwise, the correction system will not work and you will not pass the evaluation.

Introduction

Sherlock Holmes must face his arch-enemy Professor Moriarty, who has once again plotted something devilish. Moriarty's associates have kidnapped a set of people from all over the country and forced them to wear vests surrounded by explosives. Moriarty's "explosive hostages" have been hidden at different locations in the city of London, which can be represented as a 2D matrix (grid). Sherlock must discover the location of the hostages using a board (another 2D matrix), otherwise they will not survive the explosion, which will also destroy the city of London.

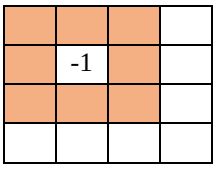
Goal:
The project has two objectives. The first goal is to help Moriarty to find the best locations for the hostages in the city. The second goal consists in helping Sherlock to discover the locations of these "explosive hostages" to save the city of London.

Exercise 1: Create the procedure *Lets_Play_Shecklocked()*
Make a procedure called *Lets_Play_Shecklocked()*, without input parameters, that displays:

```
Welcome to the game "Sherlocked". You will need to save the city of
London from Moriarty's "explosive hostages". The program will receive
as input:

   -  N, defining the size of the square grid.
   -  M, defining the number of hostages wearing explosive vests.

The program will create the NxN grid and find the place for the M
explosive hostages. Afterwards, the NxN board will be created and
Sherlock will try to find the location of the hostages by selecting
cells. Sherlock has a limited time to save the city of London from the
explosions, so he can only check a maximum of 2*M cells.

The program will generate a txt file containing the solution (the grid,
the board, and the result of the game).
```

Exercise 2: Implement the function *isValid()*

Create a function *isValid()* to check if the current position in the grid (eg. (row, col) or (x,y)) is valid for a hostage to be placed. The **condition for a valid cell position** is that two "explosive hostages" cannot be adjacent to each other. In other words, if a hostage is planted in a particular cell (see value -1), its 3x3 neighboring adjacent cells (see orange color) cannot contain any other explosive hostage. The function will return *True* if it is a valid position, *False* otherwise.



Exercise 3: Implement the function *PlantExplosives()*

Write the **recursive** function *PlantExplosives()* to find the locations (positions in the grid) for the M "explosive hostages", satisfying the constraint implemented already by the *isValid()* function. This function will make use of **backtracking** whenever the location of an explosive hostage in a particular position cannot lead to a solution. The function will return *True* if there is a solution (the M explosive hostages could be correctly placed in the grid), *False* otherwise.

Exercise 4: Implement the function *Assign_numbers_to_grid*()

Assuming that the grid could be created with the M explosive hostages correctly placed (so, function *PlantExplosives* returns *True*), write a function to glide through all the cells in the grid and assign numbers in the following manner: Assign the value "-1" to the cells that contain an explosive hostage, whereas the rest of the cells should denote how many neighboring hostages are adjacent to it.

For example, given the input grid in the left, the expected output should be:

-1				-1
		-1		
				-1
		-1		

→

-1	2	1	2	-1
1	2	-1	3	2
0	1	1	2	-1
0	1	1	2	1
0	1	-1	1	0

Here "-1" denotes the explosive hostage locations.

These numbers can help Sherlock to guess how close he is from the "explosive hostage".

Exercise 5: Implement the function *Play_Sherlock()*
The function will use the ***grid*** resulting from calling *Assign_numbers_to_grid()* and **M**, the number of hostages. It will create the *numpy* array ***board***, of the same size of ***grid***. It will be initialized with 9. Sherlock will have 2*M trials to find the locations of the explosive hostages.

The steps will be the following:

1. Ask Sherlock to provide the position (x,y) of the cell to be checked.
2. Copy the value of that cell in the **grid** to the **board**. I mean, board(x,y)=grid(x,y).
3. Print the board, so that Sherlock can visualize the result.
4. Go back to step 1 until the M hostages have been located or until the number of trials has reached 2*M. If the M hostages have been located, return *True*, otherwise, return *False*.

Exercise 6: Implement ***main()*****:**
Implement the main program that will make the calls to the corresponding functions and procedures to play the "Sherlocked" game. The sequence is as follows:

1. Call the procedure *Lets_Play_Shecklocked()* to inform about the game.
2. Ask the user to provide the following values:
   - **N**, the size of the 2D square matrix.
   - **M**, the number of explosive hostages.
3. Create the *numpy* array ***grid***, of dimension NxN, and initialize it with 0's.
4. Call the function *PlantExplosives()* to find a place for the **M** "explosive hostages", ensuring that two explosives are not adjacent to each other.
5. If the function returns *False*, print the message and exit:
   "Error: It is not possible to place M hostages in the grid".
6. If the function returns *True*, call the function *Assign_numbers_to_grid()* to assign values to all the cells of the board based on the positions where the bombs have been placed.
7. Call function *Play_Sherlock()* so that Sherlock can guess the locations of the explosives.
8. If *Play_Sherlock()* returns *True*, print the message: *"Sherlock saved London"*, otherwise, print *"London has been destroyed!"*.
9. Generate the file "solution_<N>_<M>.txt", where N and M are the values provided in step 2 (for example, if N=5 and M=8, the file will be "solution_5_8.txt"). The file will store the **grid**, the **board**, and the message (from step 8). For example:

-1	2	1	2	-1
1	2	-1	3	2
0	1	1	2	-1
0	1	1	2	1
0	1	-1	1	0
-1	9	9	9	-1
9	9	-1	3	9
9	9	9	2	-1
9	9	1	9	9
9	9	-1	9	9
Sherlock saved London!				

Important Remarks:

- The use of **numpy** array for **grid** and **board** is mandatory (do not use lists).
- The functions do not specify the number of parameters required for each function.
- Feel free to create more functions whenever necessary.
- Comments in the code are very welcome.

Delivery
Deadline: January 4, 2023.
The project will be done in pairs, and the delivery will be via *Caronte*. Instructions:

- Upload the file "Shecklocked.py".
- Only one of the members of the group must do it.
- At the beginning of the Python file, you should write, as comments, the name, surname and NIU of the members of the group.