

---

# Homework Assignment

---

04 June 2022

*Name: Dumitrescu Marian-Daniel*

*Speciality: Calculatoare Română 2nd Year*

*Group: 2.2 A*

# 1 Problem statement

Let us suppose the  $k \times k$  grid presented in Figure 2. The grid is configured with a pattern of walls. You are required to place  $n$  archers on this grid such that they cannot shoot each other. An archer can shoot up, down, left, right and also diagonally and its shoot can reach at most  $w$  locations in all directions, up to the grid edges.

For this problem we will need to input this data: the length of the grid:  $k$ ; the number of archers:  $n$ ; the strength(attack range) of the archers:  $w$ ; and finally the initial grid. The walls are also part of the archer's grid and their position does not change. The final position of the archers must be determined so that they will not attack each other.

We will consider some examples to see how the problem should be resolved:

- Example 1:

Input				Output			
k		4					
n		3					
w		2					
.	.	W	.	A	.	W	A
W	.	.	.	W	.	.	.
.	.	W	.	A	.	W	.
.	W	.	.	.	W	.	.

- Example 2:

Input				Output			
k		4					
n		4					
w		7					
.	W	.	W	A	W	A	W
W	.	.	.	W	.	.	.
.	W	.	.	.	W	.	A
.	W	W	.	A	W	W	.

The problem proposed is a N-Queens Problem in which we can place walls that the supposed queens (in our case archers,... ) cannot attack through these walls. Also, the queens can have an attack range (the number of cells that they can attack from). This problem could be resolved with Backtracking, Depth-First Search, and Breadth-First Search, but we will use just the Depth-First Search algorithm to determine if it exists a solution, and if it exists, we will export it accordingly.

## 2 Pseudocode of Algorithms

```

function DFS(grid,  $n$ ,  $k$ ,  $w$ , row = 0, col = 0)
  if  $n = 0$  then
    return true
  end if
  if col =  $k$  then
    row  $\leftarrow$  row + 1
    col  $\leftarrow$  0
    if row =  $k$  then
      return false
    end if
  end if
  if IS_VALID(grid,  $k$ ,  $w$ , row, col) then
    grid[row][col]  $\leftarrow$  'A'
    if DFS(grid,  $n - 1$ ,  $k$ ,  $w$ , row, col + 1) then
      return true
    end if
    grid[row][col]  $\leftarrow$  '.'
  end if
  return DFS(grid,  $n$ ,  $k$ ,  $w$ , row, col + 1)
end function

function IS_VALID(grid,  $k$ ,  $w$ , row, col)
  if row < 0 or col < 0 or row  $\geq k$  or col  $\geq k$  then
    return false
  end if
  if IS_ARCHER(grid, row, col) then
    return false
  end if

```

```

if IS_WALL(grid, row, col) then
    return false
end if
if not CHECK_HORIZONTAL(grid,  $k$ ,  $w$ , row, col) then
    return false
end if
if not CHECK_VERTICAL(grid,  $k$ ,  $w$ , row, col) then
    return false
end if
if not CHECK_MAIN_DIAGONAL(grid,  $k$ ,  $w$ , row, col) then
    return false
end if
if not CHECK_SECONDARY_DIAGONAL(grid,  $k$ ,  $w$ , row, col) then
    return false
end if
return true
end function

function CHECK_HORIZONTAL(grid,  $k$ ,  $w$ , row, col)
    if col = 0 then
        return true
    end if
    start_col  $\leftarrow$  col - 1
    temp_col  $\leftarrow$  start_col
    while  $w > 0$  and temp_col  $\geq 0$  do
        if IS_WALL(grid, row, temp_col) then
            return true
        end if
        if IS_ARCHER(grid, row, temp_col) then
            return false
        end if
        temp_col  $\leftarrow$  temp_col - 1
         $w \leftarrow w - 1$ 
    end while
    return true
end function

function CHECK_VERTICAL(grid,  $k$ ,  $w$ , row, col)
    if row = 0 then

```

```

        return true
    end if
    start_row  $\leftarrow$  row - 1
    temp_row  $\leftarrow$  start_row
    while  $w > 0$  and temp_row  $\geq 0$  do
        if IS_WALL(grid, temp_row, col) then
            return true
        end if
        if IS_ARCHER(grid, temp_row, col) then
            return false
        end if
        temp_row  $\leftarrow$  temp_row - 1
         $w \leftarrow w - 1$ 
    end while
    return true
end function

function CHECK_MAIN_DIAGONAL(grid,  $k$ ,  $w$ , row, col)
    if row = 0 or col = 0 then
        return true
    end if
    start_row  $\leftarrow$  row - 1
    start_col  $\leftarrow$  col - 1
    temp_row  $\leftarrow$  start_row
    temp_col  $\leftarrow$  start_col
    while  $w > 0$  and temp_row  $\geq 0$  and temp_col  $\geq 0$  do
        if IS_WALL(grid, temp_row, temp_col) then
            return true
        end if
        if IS_ARCHER(grid, temp_row, temp_col) then
            return false
        end if
        temp_row  $\leftarrow$  temp_row - 1
        temp_col  $\leftarrow$  temp_col - 1
         $w \leftarrow w - 1$ 
    end while
    return true
end function

```

```

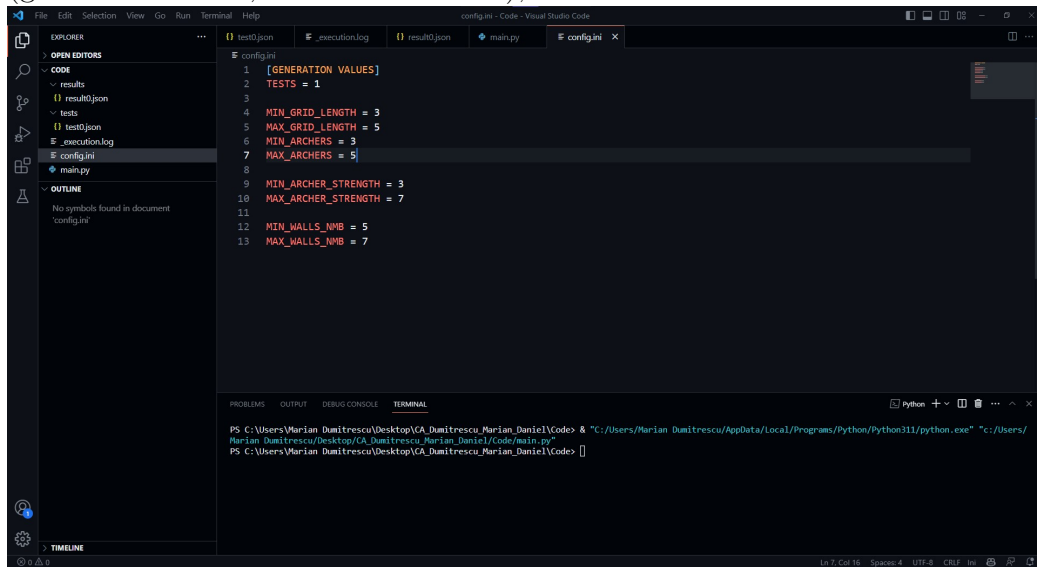
function CHECK_SECONDARY_DIAGONAL(grid,  $k$ ,  $w$ , row, col)
  if row = 0 or col =  $k - 1$  then
    return true
  end if
  start_row  $\leftarrow$  row - 1
  start_col  $\leftarrow$  col + 1
  temp_row  $\leftarrow$  start_row
  temp_col  $\leftarrow$  start_col
  while  $w > 0$  and temp_row  $\geq 0$  and temp_col  $< k$  do
    if IS_WALL(grid, temp_row, temp_col) then
      return true
    end if
    if IS_ARCHER(grid, temp_row, temp_col) then
      return false
    end if
    temp_row  $\leftarrow$  temp_row - 1
    temp_col  $\leftarrow$  temp_col + 1
     $w \leftarrow w - 1$ 
  end while
  return true
end function

```

### 3 Application outline

To resolve this problem I used the Depth-First Search algorithm. Depth-First Search is used to search graphs by going through the largest possible paths (the deepest that could be). The algorithm will check for every cell in the grid if we can place an archer using the IS\_VALID function. If there are no archers remaining to place in a function call for the DFS function, we can assume that we found a solution! This application is structured in three functionalities:

- Fetching for the configuration values for the test generation from 'execution.log' (generation limits, number of tests...);



The screenshot shows the Visual Studio Code interface. The Explorer sidebar on the left shows a project structure with folders 'results' and 'tests', and files 'result0.json', 'test0.json', 'execution.log', 'config.ini', and 'main.py'. The 'config.ini' file is open in the editor, showing the following content:

```
1 [GENERATION VALUES]
2 TESTS = 1
3
4 MIN_GRID_LENGTH = 3
5 MAX_GRID_LENGTH = 5
6 MIN_ARCHERS = 3
7 MAX_ARCHERS = 5
8
9 MIN_ARCHER_STRENGTH = 3
10 MAX_ARCHER_STRENGTH = 7
11
12 MIN_WALLS_NMB = 5
13 MAX_WALLS_NMB = 7
```

The Terminal window at the bottom shows a PowerShell command being executed:

```
PS C:\Users\Marian Dumitrescu\Desktop\CA_Dumitrescu_Marian_Daniel\Code> & "C:/Users/Marian Dumitrescu/AppData/Local/Programs/Python/Python311/python.exe" "c:/Users/Marian Dumitrescu/Desktop/CA_Dumitrescu_Marian_Daniel/Code/main.py"
PS C:\Users\Marian Dumitrescu\Desktop\CA_Dumitrescu_Marian_Daniel\Code>
```

- We will test generated values (and then export the initial values for the test) for the proposed problem and we will represent the solutions (both exported using *json* file notation);

The screenshot shows the Visual Studio Code editor with two files open: `test0.json` and `result0.json`. The `test0.json` file contains a JSON object with the following structure:

```

{
  "length": 4,
  "archers": 4,
  "strength": 7,
  "grid": [
    [0, 0, 0, 0],
    [0, 0, 0, 0],
    [0, 0, 0, 0],
    [0, 0, 0, 0]
  ]
}

```

The `result0.json` file contains a similar JSON object, but with a `solution` field instead of `grid`:

```

{
  "length": 4,
  "archers": 4,
  "strength": 7,
  "solution": [
    [0, 0, 0, 0],
    [0, 0, 0, 0],
    [0, 0, 0, 0],
    [0, 0, 0, 0]
  ]
}

```

The Explorer sidebar on the left shows the project structure, including `tests`, `result0.json`, `main.py`, and `config.ini`. The Terminal at the bottom shows the command used to run the program:

```

PS C:\Users\Marian Dumitrescu\Desktop\CA_Dumitrescu_Marian_Daniel\Code> & "C:/Users/Marian Dumitrescu/AppData/Local/Programs/Python/Python311/python.exe" "C:/Users/Marian Dumitrescu/Desktop/CA_Dumitrescu_Marian_Daniel/Code/main.py"
PS C:\Users\Marian Dumitrescu\Desktop\CA_Dumitrescu_Marian_Daniel\Code>

```

- Running the proposed problem and logging our execution of the program while printing some useful visualizations for the tests and tracking the time elapsed for the said program.

The screenshot shows the Visual Studio Code editor with the `execution.log` file open. The log contains the following text:

```

1 The values for test no. 0 were generated in 0.66 milliseconds!
2 For test no. 0 @ C:\Users\Marian Dumitrescu\Desktop\CA_Dumitrescu_Marian_Daniel\Code\tests\test0.json
3 k : 4, n : 4, w : 7
4 The solution for test no. 0 was computed in 0.03 milliseconds!
5 [0, 0, 0, 0] [0, 0, 0, 0]
6 [0, 0, 0, 0] [0, 0, 0, 0]
7 [0, 0, 0, 0] [0, 0, 0, 0]
8 [0, 0, 0, 0] [0, 0, 0, 0]
9
10 The program ended successfully in 1.47 milliseconds!
11

```

The Explorer sidebar on the left shows the project structure, including `tests`, `result0.json`, `main.py`, and `config.ini`. The Terminal at the bottom shows the command used to run the program:

```

PS C:\Users\Marian Dumitrescu\Desktop\CA_Dumitrescu_Marian_Daniel\Code> & "C:/Users/Marian Dumitrescu/AppData/Local/Programs/Python/Python311/python.exe" "C:/Users/Marian Dumitrescu/Desktop/CA_Dumitrescu_Marian_Daniel/Code/main.py"
PS C:\Users\Marian Dumitrescu\Desktop\CA_Dumitrescu_Marian_Daniel\Code>

```



The application was written in Python, because I wanted to get myself acquainted with it, thinking that it would be a great experience writing the code for the assignment while learning itself!

I used the *randint* function from the Python's *random* module to generate the random values, *configparser* module to get the configuration values for the program, Python's *os* module to get some useful os-centric functions and Python's *json* module to export the inputs and outputs for the program for later use, if we would like to expand the application with a Graphical User Interface in the near future. To generate new tests, run the main.py script, and if you want to change some configuration values for the program, edit 'config.ini'.

## 4 Conclusion

This homework assignment was a great way to learn more about uninformed search problems and a great way to learn to use the Python programming language as an important tool to define new and interesting ideas using software. I even learned a lot more about L<sup>A</sup>T<sub>E</sub>X than last year. :)

## 5 References

- <https://www.tablesgenerator.com/> very useful to make the examples!
- <https://en.wikipedia.org/wiki/JSON>
- <https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/>
- <https://www.learnlatex.org/en/>
- <https://www.youtube.com/> for great latex tutorials :P