INFORMATICS INSTITUTE OF TECHNOLOGY

Foundation Certificate for Higher Education

(Foundation September 2023)

Module Name: DOC 334 Computer Programming

Module leader: Mr. Nishan Saliya Harankahawa

Assignment type: Individual coursework

Group ID: I1

Submission date: 21.02.2024

Student Id	Student Name
20231259	Mr. H. I. A. Haneef

Abstract

This report covers the percolation problem and it solution program which is the use of a Python program to simulate and analyze the percolation process in geology. Percolation, the movement of water through soil and permeable rocks, is a complex process often studied through computer simulations due to the difficulty of obtaining exact results from analytical models. The program generates a grid filled with random numbers and empty spaces, representing the randomness inherent in percolation. Each column in the grid represents a percolation process, with successful percolation indicated by an “Ok” and unsuccessful percolation due to blockages which is the empty spaces indicated by a “NO”. This program simplifies the exploration of percolation dynamics, providing valuable insights into this crucial geological process.

Also, it covers the algorithm used to create the program, test cases, screenshots of the program outputs in various states with full programs codes are included in the report.

Acknowledgement

I would like to express my deepest appreciation to Mr. Nishan Saliya, our respected lecturer and module leader for his constant guidance and clarifications throughout the assignment. I want to give a special thanks to our tutorial lecturer Ms. Shafka Fuard for her engaging tutorials and the clarifications of problems which were given in the tutorials which made it possible for me to improve my programming skills for the making this project.

In addition, I am profoundly grateful to my family and friends for their continuous encouragement and backing, which were crucial in the successful completion of this assignment.

Table of Contents

Lists of figures

List of tables

1. Introduction

1.1. Problem:

In geology field percolation process is used to filtrate water through soil and permeable rocks for the water to flow and recharge the groundwater in the water table and aquifers. Due to the complexity involved in obtaining exact results from analytical models of percolation, computer simulations are typically used to interpret the outcome. The similarity between the random empty spaces in a program and the percolation process is randomness. (Spielmaker, 2024)

In the grid generation, randomness determines which cells are empty, while in the percolation process, it determines the connectivity of the medium and whether percolation occurs.

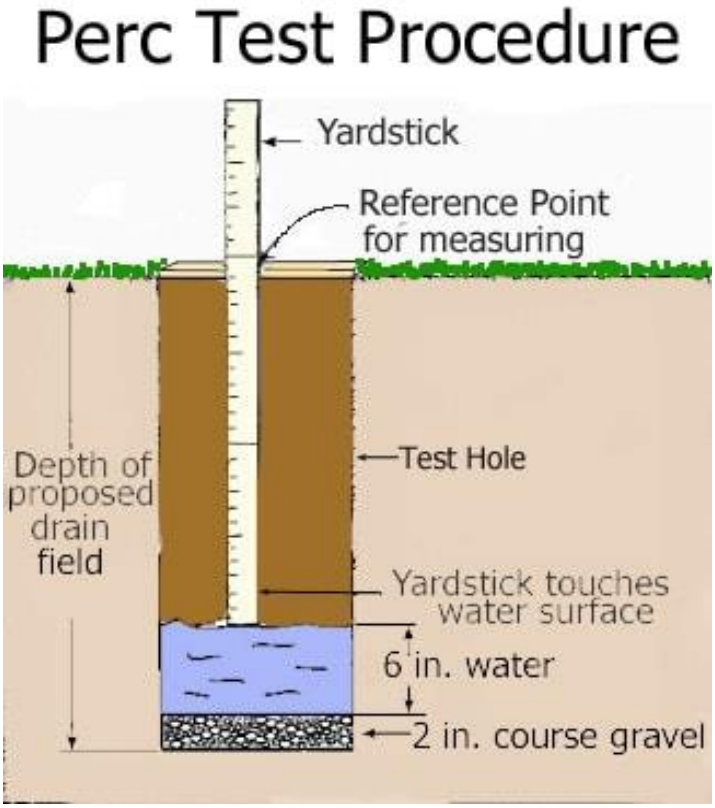

Figure 1: water percolation.

1.2. Solution

A python program is developed to determine the outcome of the percolation process. The program produces a grid populated with random 2-digit numbers with random empty spaces based on the given probability which is left inside the grid where each column in the grid acts as the process of percolation. Ensuring the pathway from top to bottom is unhindered. If the percolation of water can be achieved successfully “Ok” is displayed at the end of each column and if the percolation is hindered due to the spaces in the column “NO” is displayed and analyzed to see the outcome whether the percolation can be done or not.

Moreover, the maximum number of percolation processes are determined by the user with a maximum of 9 processes and minimum of 3 processes can be generated in a grid, which is basically the number of columns of the grid. Additionally, the program will create a html and text file for each grid creation with its date of creation and random unique number as its file name for future reference of the possibilities of percolation processes. This is created in a way where there won’t be any errors and errors are managed to create a flawless working program. This is to create a clear understanding and visualization of the percolation process and estimation of possibilities connected with real life scenarios such as the above-mentioned problem percolation of water through soil and permeable rocks.

1.3. Solution algorithm

1.3.1. Algorithm for the main file(perc.py)

1. Import Required Modules:
    - ❖ Import the random module for generating random numbers.
    - ❖ Import the date module from the datetime library to handle dates.
    - ❖ Import PrettyTable for creating a formatted table.
    - ❖ Import sys for handling command-line arguments.
2. Define Functions:
    - ❖ creating_grid_and_insertion(columns, rows, probability):
        - Initialize an empty grid list using for loop where the number of inner lists is determined by passed parameter rows and the number of elements

in each list are determined by the passed parameter columns which is named as grid.

- Initialize a list to track percolation where ‘OKs are inserted according to the number of columns. The list is named check_percolation.
- Iterate over rows and columns to fill the grid randomly with 2-digit integers or empty strings based on a given probability.
- Update the percolation check list with ‘no’ based on the presence of empty strings in each column.
- Return the grid and percolation check list.

❖ printing_grid(grid, check_percolation):

- Creates a string representation from the passed parameter grid in a for loop where each element are separated by tab character and stored inside a variable called full_grid in each loop occurrence and in the percolation check list is represented in a string format and where each elements are separated by tab character which is then added into the full_grid variable as well.
- Initializing a variable called table where the grid using PrettyTable is created in a for loop where the lists in each row is inserted into it and appending the percolation check list which was made into one string and each element separated by tab character at the bottom is printed along with the table being above it.

❖ create_text_file_and_html_file(full_grid, table, check_percolation):

- Generate a filename based on the current date and a random 4-digit number.
- Write the grid data which is the full_grid variable to a text file with the generated filename.
- Write the grid data which is the table along with the percolation check list turned into a string then each element is separated by tab character in the bottom of the table is inserted into an HTML file with the generated filename.

❖ execution(columns, rows):

- Call creating_grid_and_insertion() to create the grid and percolation check list.
- Print the grid using printing_grid().
- Create text and HTML files using create_text_file_and_html_file().

❖ Main Function (main()):

- Handles terminal-line arguments
- If no arguments are provided, generate a default 5x5 grid.
- If one argument is provided in the format "RxC", where R and C are integers, generate a grid with R rows and C columns.
- If the grid size exceeds 9x9 or is below 3x3, print an error message saying, ‘grid size exceeds 9x9 or below 3x3 therefore try again’.
- Error Handling:
- Used try-except blocks to catch any exceptions during execution and print appropriate error message as wrong format.

3.Execute the Main Function:

❖ Call the main() function to start the program execution.

1.4. Screenshots of the results in various outcomes

Perc.py:

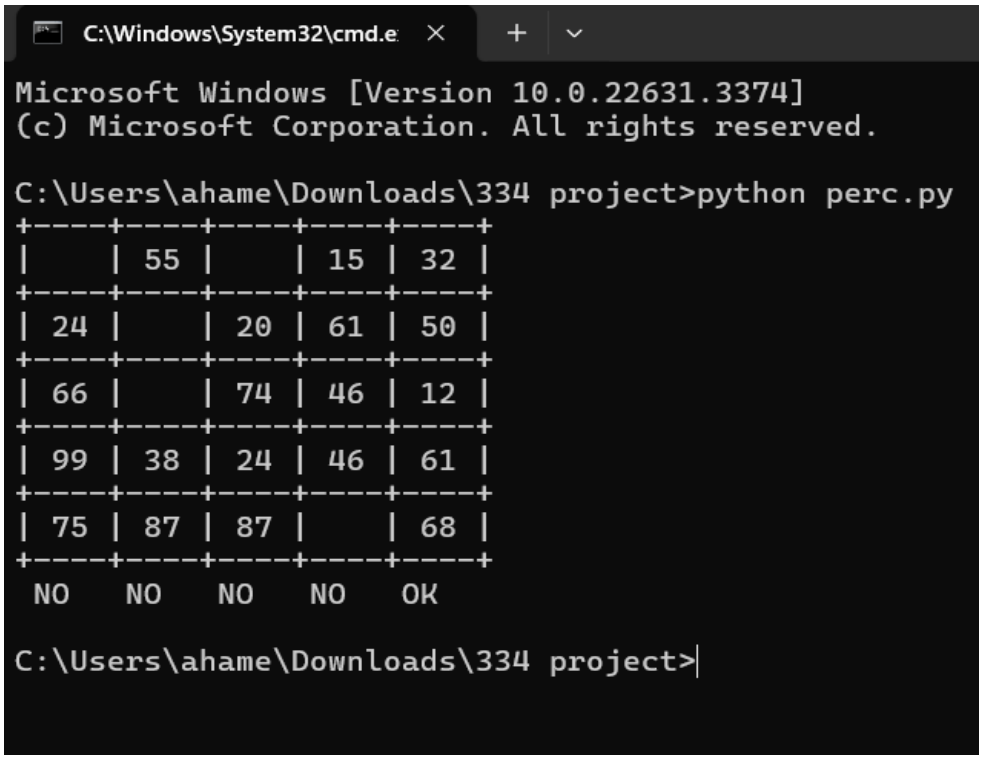

Figure 2:terminal output.

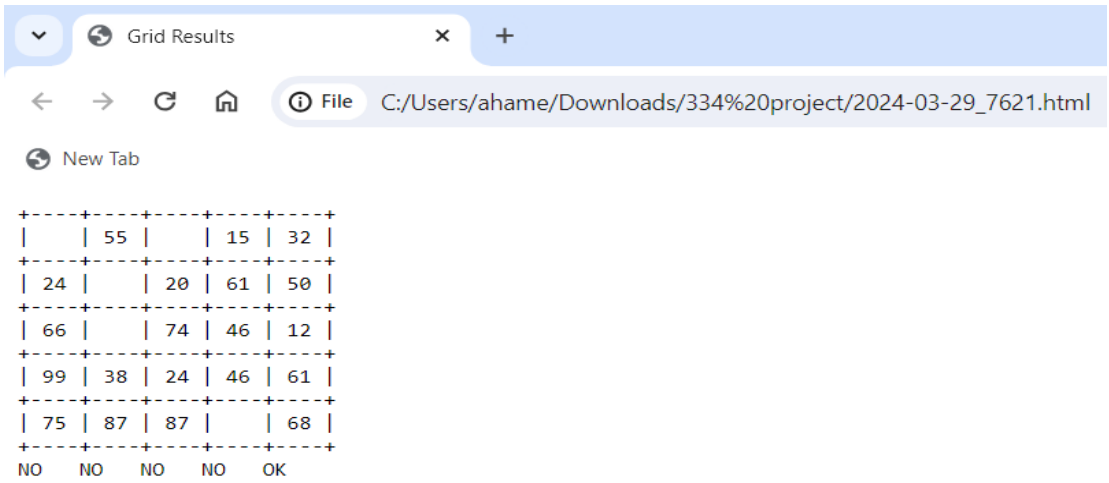

Figure 3:html output.

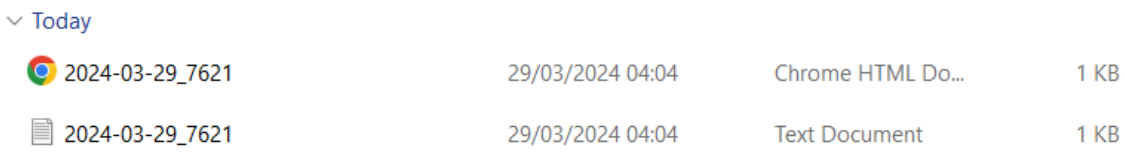

Figure 4: html and text file creation.

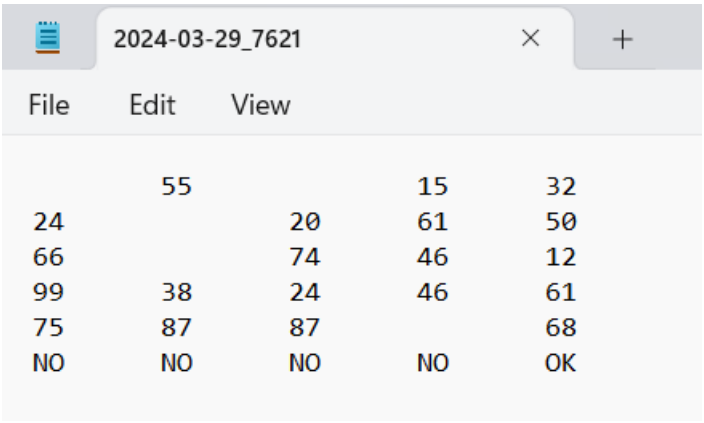

Figure 5: text file output.

1.5. Program codes.

1.5.1. Main program(perc.py)

```
#importing modules
import sys
import percolation_functions


#creating main function
def main():
    #using try function to check for errors and handle errors
    try:
        #using the condition if the length of the terminal argument is 1 and the command is perc.py it will call in the execution function from the module and create a 5x5 percolation grid or else it will give a wrong statement prompt
        if len(sys.argv) == 1:
            if sys.argv[0]=='perc.py':
                rows = 5
```

```
            columns = 5
            percolation_functions.execution(columns, rows)
        else:
            print('wrong statement')
    #using the condition if the length of the terminal argument is 2 and according to the second argument the length of the column and rows are determined with a maximum being 9 and minimum being 3 for the rows and column values and will call in the module function to print the grid or else it will print wrong format
    elif len(sys.argv) == 2:
        rows, columns = map(int, sys.argv[1].split('x'))
        if 3 <= rows <= 9 and 3 <= columns <= 9:
            percolation_functions.execution(columns, rows)
    #or else giving an output as the grid size entered being above 9x9 or below 3x3
        else:
            print("grid size exceeds 9x9 or below 3x3 therefore try again")
    #if the length of the system argument being above 2 or 0 ,a message is printed as wrong format
    else:
        print("Wrong format")
        return

except:
    print('wrong format')

main()#calling in the main function for the running of the code
```

1.5.2. module program file(percolation_functions)

```
#importing modules

import random #importing random for randomly choosing algorithm

from datetime import date #importing date from datetime module for name the text file

from random import randint #importing randint from random randomly choosing between numbers algorith to populate the grid

from prettytable import PrettyTable #importing pretty table to prettify the table




#creating an execution function to run all the defined functions

def execution(columns,rows):

    grid, check_percolation = creating_grid_and_insertion(columns, rows, probability=0.15)

    full_grid = printing_grid(grid, check_percolation)

    table = PrettyTable(header=False,hrules=True)

    for row in grid:

        table.add_row(row)

    create_text_file_and_html_file(full_grid, table,check_percolation)



#defining function for creating the full grid filled with random spaces and random 2 digit integers with the percolation process check by calling in the system argument for the number of rows

def creating_grid_and_insertion(columns,rows,probability=0.15):

    #creating a local variable called grid where it creates multiple lists within a list where the number of inner lists are determined by rows and number of elements in each inner list is determined by columns

    grid=[[" for i in range(columns)]for i in range(rows)]

    #creating a local variable to insert the ok's in a list according to the number of columns

    check_percolation = ['OK' for i in range(columns)]


    #inserting random 2 digits numbers into the grid and changing the 'ok's in percolation check list to 'no' if the grid has string values in it
```

```
    for r in range(rows):
      for c in range(columns):
        if random.random()> probability:
          grid[r][c]=randint(10,99)
        else:
          grid[r][c]=''


          check_percolation[c] = 'NO'
    return grid,check_percolation



def printing_grid(grid,check_percolation):
    #creating a variable called full grid for transfering the final grid including perculation check into it
    full_grid=''
    #creating variable which acts as the pretty table function
    table=PrettyTable(header=False,hrules=True,field_names=False)
    #using for loop each inner_lists are made into a string from list and each of the elements in the string are seperated by tabs and assigned into the full grid vairiable
    for r in grid:
      full_grid+= "\t".join(map(str, r)) +'\n'
      #each inner_lists are add in a row and prettified in a table
      table.add_row(r)


    #the check_perculation list is added into the full grid variable to create the whole table with perculation check
    full_grid+="\t".join(check_percolation)
    #the prettified table is printed with check perculation list under it where it becomes one string and each element are seperated by spaces to display a final table with perculation status
    print(table,'\n','   '.join(check_percolation))
    return full_grid
```

```
def create_text_file_and_html_file(full_grid,table,check_percolation):


    #creating a local variable for naming the date today for naming of the text file

    date_today=date.today()

    #creating a local variable for creating a random 4 digit number for naming of the text file

    four_digit_numbers=random.randint(1000,9999)

    #creating the whole file name using the created local variables the date and random 4 digit number

    file_name=(f'{date_today}_{four_digit_numbers}.txt')

    #Creating a text file with the file_name variable as its name and writing the created whole grid variable into the text file

    with open(file_name , 'w') as fn:

      fn.write(f'{full_grid}')

      fn.close()



#creating html code by opening the file as it's random 4 digit number and date today as its name

    html_f=(f'{date_today}_{four_digit_numbers}.html')

    with open(html_f, "w") as html_file:

      html_file.write("<html>\n")

      html_file.write("<head><title>Grid Results</title></head>\n")

      html_file.write("<body>\n")

      html_file.write("<pre>\n")

      html_file.write(table.get_string()+'\n'+'   '.join(check_percolation))

      html_file.write("</pre>\n")

      html_file.write("<p>"   "</p>\n")

      html_file.write("</body>\n")

      html_file.write("</html>\n")

      return date_today,four_digit_numbers,full_grid
```

2. Test cases and their outcomes.

Test case No.	Input	Expected Outcome	Actual Outcome	Result
1	When user Input's perc.py in the terminal	Creates a default grid of 5x5 with 2-digit numbers in them with random empty spaces. Along with percolation status And saves them in a text file and html file	As Expected,	Pass
2	When user Input's perc.py 4x8 in the terminal	Creates a grid of 4x8 with 2-digit numbers in them with random empty spaces along with percolation status And saves them in a text file and html file	As Expected,	Pass
3	When user Input's perc.py 2x3 in the terminal	Displays 'grid size exceeds 9x9 or below 3x3 therefore try again'	As Expected,	Pass
4	When user Input's perc.py 10x5 in the terminal	Displays 'grid size exceeds 9x9 or below 3x3 therefore try again' message	As Expected,	Pass
5	When user Input's perc.py 4x5 ad in the terminal	Displays "wrong format"	As Expected,	Pass
6	When user Input's perc.py 3@4 in the terminal	Displays "wrong format"	As Expected,	Pass
7	When user input's perc.py 5x5 in the terminal	Creates a grid of 7x7 with 2-digit numbers in them with random empty spaces and saves them in a text file and html file	As Expected,	Pass
8	When user input's Perc.py 9x8	Creates a grid of 9x9 with 2-digit numbers in them with random empty spaces and saves them in a text file and html file	As Expected,	Pass

Table 1: test cases

2.1. Test case 1 outcomes

```
C:\Users\ahame\Downloads\334 project>python perc.py
+----+----+----+----+----+
| 21 | 70 | 74 | 43 | 60 |
+----+----+----+----+----+
|    | 43 | 86 | 11 | 69 |
+----+----+----+----+----+
| 15 | 95 | 55 | 74 | 81 |
+----+----+----+----+----+
|    |    | 83 | 38 | 70 |
+----+----+----+----+----+
| 59 | 33 | 35 | 13 | 19 |
+----+----+----+----+----+
 NO   NO   OK   OK   OK
```

Figure 6:test case 1 terminal outcome.

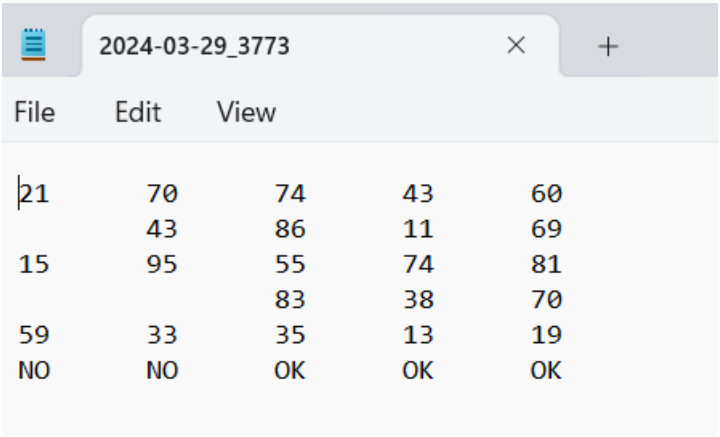

Figure 7:test case 1 text file.

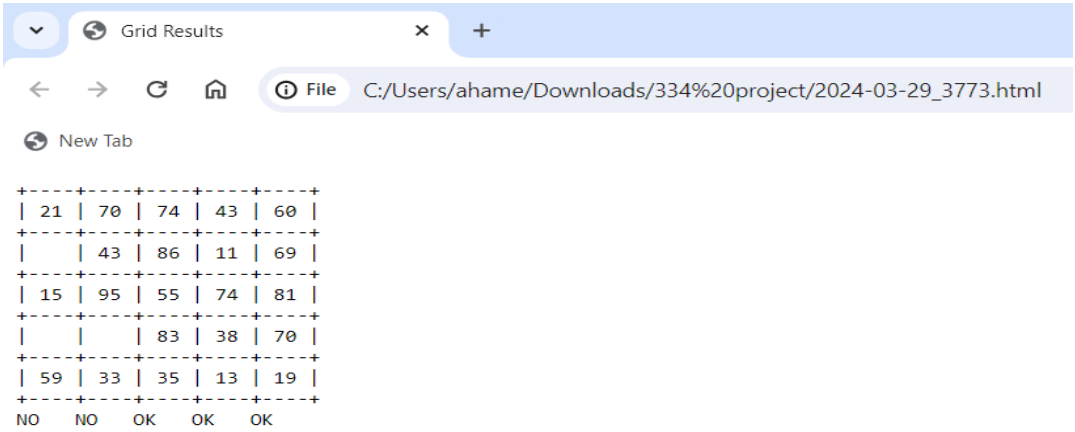

Figure 8:test case 1 html file.

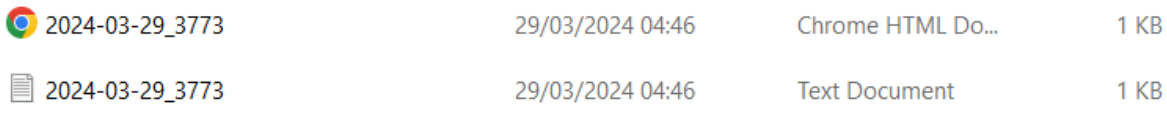

Figure 9: test case 1 text and html files creations

2.2. Test case 2 outcomes

```
C:\Users\ahame\Downloads\334 project>python perc.py 4x8
+----+----+----+----+----+----+----+----+
| 84 | 67 | 85 | 79 | 82 | 98 | 97 | 80 |
+----+----+----+----+----+----+----+----+
| 96 |    | 89 | 40 | 43 | 11 | 85 | 21 |
+----+----+----+----+----+----+----+----+
|    | 19 | 35 | 82 | 94 |    | 35 | 59 |
+----+----+----+----+----+----+----+----+
| 55 | 23 | 97 | 27 | 78 | 33 |    | 40 |
+----+----+----+----+----+----+----+----+
 NO   NO   OK   OK   OK   NO   NO   OK
```

Figure 10 : test case 2 outcome.

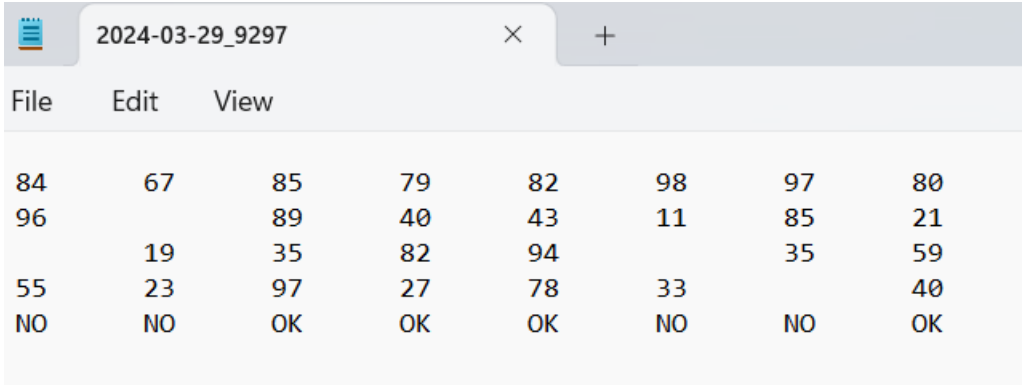

Figure 12: test case 2 text file.

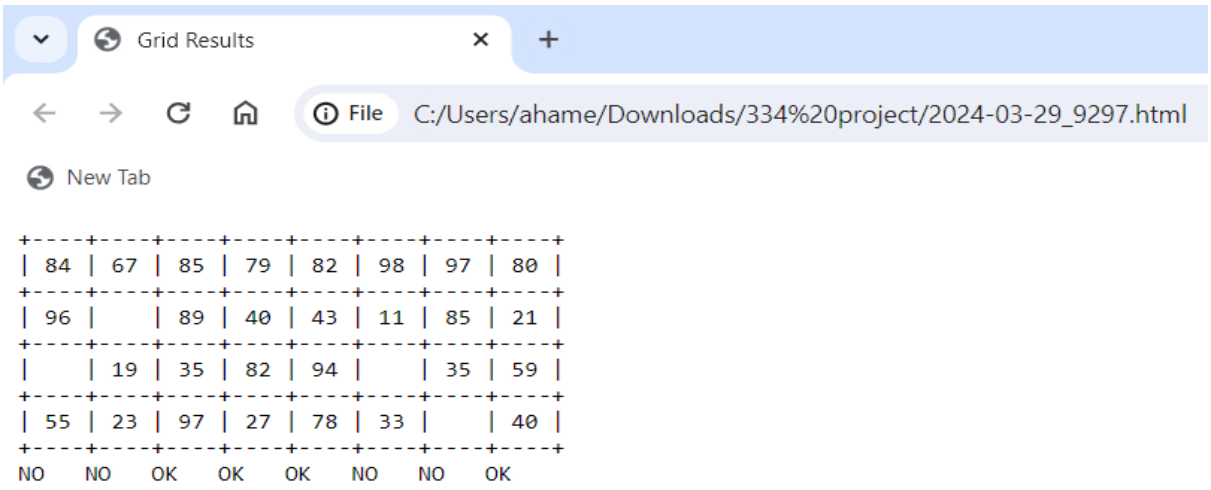

Figure 11:test case 2 html file.

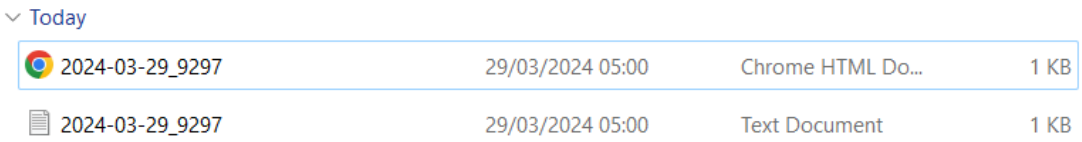

Figure 13: test case 2 text and html files creation.

2.3. Test case 3 outcomes

```
C:\Users\ahame\Downloads\334 project>python perc.py 2x3
grid size exceeds 9x9 or below 3x3 therefore try again
```

Figure 14: test case 3 terminal outcome.

2.4. Test case 4 outcomes

```
C:\Users\ahame\Downloads\334 project>python perc.py 10x5
grid size exceeds 9x9 or below 3x3 therefore try again
```

Figure 15: test case 4 terminal outcome.

2.5. test case 5 outcomes

```
C:\Users\ahame\Downloads\334 project>python perc.py 4x5 ad
Wrong format
```

Figure 16: test case 5 terminal outcome.

2.6. Test case 6 outcomes

```
C:\Users\ahame\Downloads\334 project>python perc.py 4@5
wrong format
```

Figure 17 : test case 6 terminal outcome.

2.7. Test case 7 outcomes

```
C:\Users\ahame\Downloads\334 project>python perc.py 5x5
+----+----+----+----+----+
| 17 | 65 |    | 88 |    |
+----+----+----+----+----+
| 18 | 17 | 43 | 65 | 38 |
+----+----+----+----+----+
| 23 | 63 | 31 |    | 27 |
+----+----+----+----+----+
| 74 | 51 | 74 | 54 | 33 |
+----+----+----+----+----+
| 53 | 45 |    | 61 | 61 |
+----+----+----+----+----+
 OK   OK   NO   NO   NO
```

Figure 18: test case 7 terminal outcome.

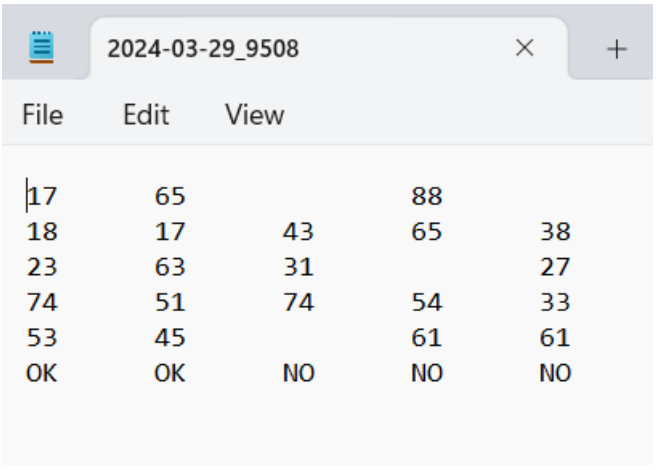

Figure 19:test case 7 text file.

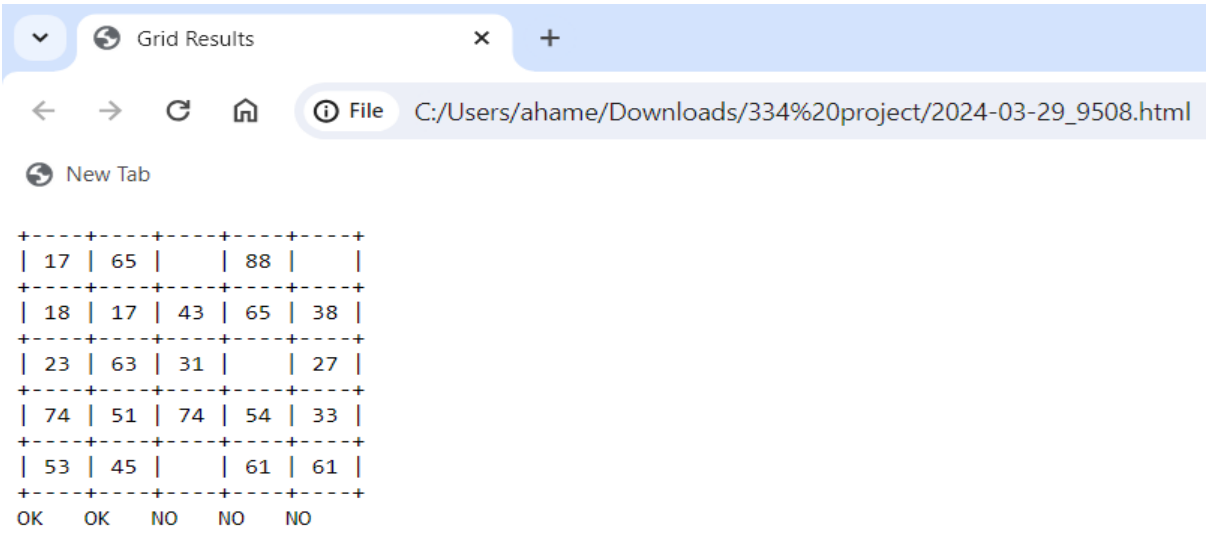

Figure 20: test case 7 html file.

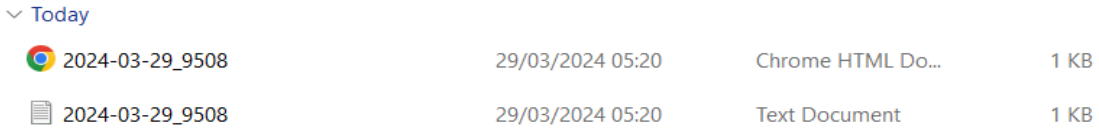

Figure 21: test case 7 html and text files creation.

2.8. Test case 8 outcomes

```
C:\Users\ahame\Downloads\334 project>python perc.py 9x9
+----+----+----+----+----+----+----+----+----+
| 96 | 26 | 87 |    | 82 | 82 |    |    | 25 |
+----+----+----+----+----+----+----+----+----+
| 78 | 48 | 38 | 71 | 49 | 97 | 31 | 68 | 31 |
+----+----+----+----+----+----+----+----+----+
| 68 | 55 | 68 | 92 | 71 | 65 | 23 | 62 | 12 |
+----+----+----+----+----+----+----+----+----+
| 32 | 24 | 30 | 90 | 87 | 24 | 34 | 37 | 66 |
+----+----+----+----+----+----+----+----+----+
| 25 | 76 | 80 | 18 | 38 | 55 | 19 | 84 | 18 |
+----+----+----+----+----+----+----+----+----+
| 71 | 69 |    | 81 | 66 |    | 22 | 34 |    |
+----+----+----+----+----+----+----+----+----+
|    | 24 |    | 58 | 48 | 22 | 21 | 71 | 59 |
+----+----+----+----+----+----+----+----+----+
| 98 | 75 | 61 | 51 | 98 |    |    | 69 | 62 |
+----+----+----+----+----+----+----+----+----+
| 29 | 65 | 29 | 13 | 60 |    | 48 | 11 | 48 |
+----+----+----+----+----+----+----+----+----+
 NO   OK   NO   NO   OK   NO   NO   NO   NO
```

Figure 22: test case 8 terminal outcome.

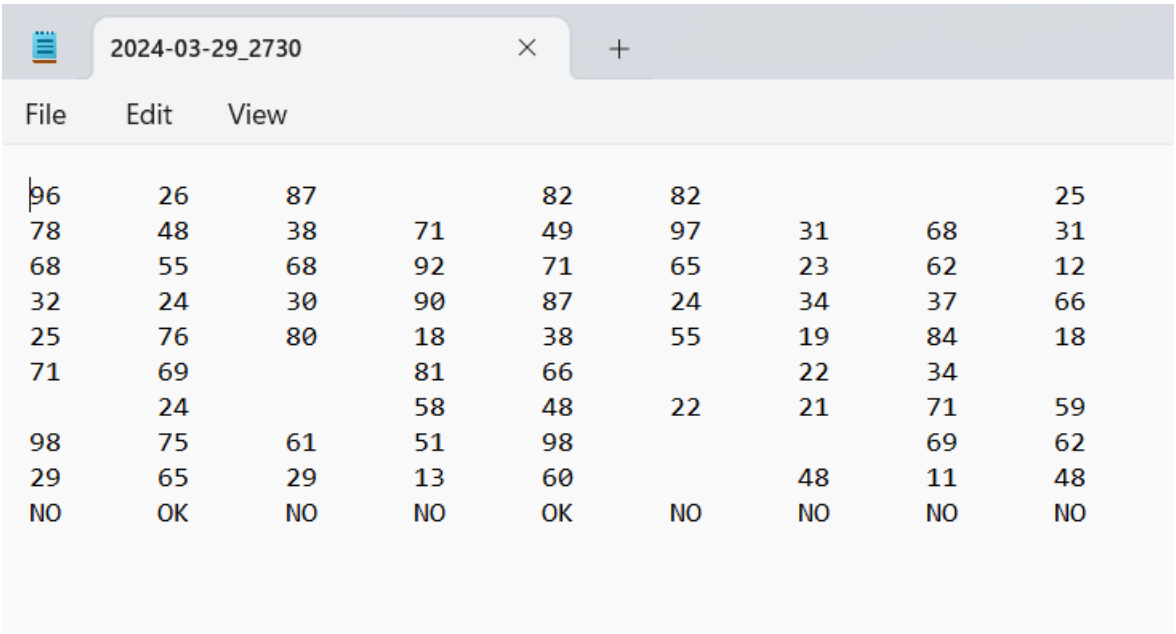

Figure 23: test case 8 text file.

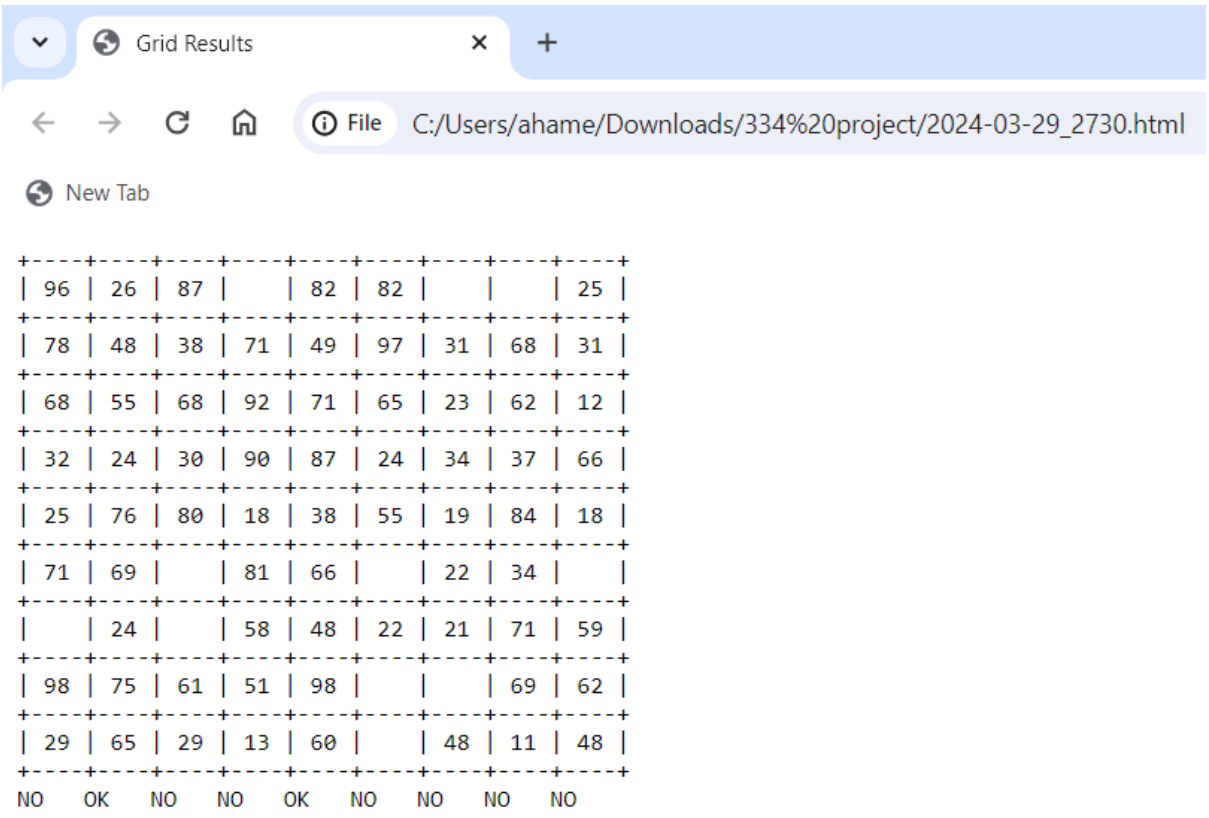

Figure 24: test case 8 html file.

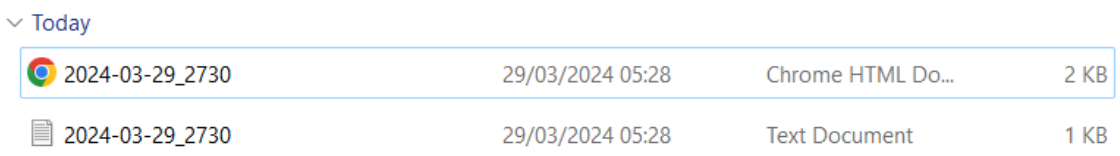

Figure 25: test case 8 text and html file creation.

3. Conclusion

The provided Python script is a flexible tool that creates grids filled with random numbers and spaces. This allows for the study of percolation within these grids. The script starts by setting up variables, importing necessary modules, and defining functions. These functions are used for creating the grid, evaluating percolation, showing the status, and saving the results in different formats. This tool makes it easier to explore various grid setups and analyse how percolation occurs within them.

References

Parewa Labs Pvt. Ltd, 2011. *programiz.* [Online]
Available at: https://www.programiz.com/html
[Accessed 25 03 2024].

Refsnes Data, 1999. *w3schools.* [Online]
Available at: https://www.w3schools.com/python/default.asp
[Accessed 25 03 2024].

Sanchhaya Education Private Limited, 2024. *GeeksforGeeks.* [Online]
Available at: https://www.geeksforgeeks.org/python-programming-language/
[Accessed 25 03 2024].

Spielmaker, D., 2024. *National agriculture in classroom.* [Online]
Available at: https://agclassroom.org/matrix/lesson/147/
[Accessed 29 3 2024].