Faculty of Computers and Data Sciences

Cyber Security department

Level 2

جامعة الإسكندرية
ALEXANDRIA
UNIVERSITY
كلية الحاسبات وعلوم البيانات

A report about:

Minesweeper game and its implementation in Python

Prepared by:

Omar Ibrahim Ahmed	**2206209**
Youssef Tamer Mohamed	**2206208**
Marwan Gaber Ramadan	**2206167**
Hadir Amr Mahmoud	**22010450**
Youssef Mohamed Fathy(AI)	**22010312**

Under supervision of:

Dr Lamiaa Aly

Introduction to AI

Fall 2024

Minesweeper is a classic single-player puzzle game that has gained popularity since its inclusion in Microsoft Windows operating systems. The game's objective is to clear a rectangular grid or board containing hidden mines without detonating any of them. The game was first introduced in the 1960s and has since become a standard inclusion in various computer operating systems.

1] Game Components

Grid/Board: The game is played on a grid, usually rectangular, consisting of cells. Each cell can either be empty, contain a number, or hide a mine.

Mines: Mines are strategically placed within the grid, and players must avoid clicking on them. When a mine is clicked, the game ends, and the player loses.

Numbers: Cells without mines are usually labeled with a number indicating the total number of mines in the adjacent cells (including diagonals). This number helps players deduce the locations of mines.

Flags: Players can place flags on cells they suspect to contain mines. This is a crucial strategy to mark potential mine locations and avoid accidental clicks.

2] Gameplay:

Starting the Game: The game begins with the player uncovering an initial cell. This first cell is typically free of mines.

Numbered Cells: When a cell is uncovered, it reveals either an empty space or a number. The number represents how many mines are in the neighboring cells.

Empty Spaces: If a player uncovers an empty cell, the game automatically reveals adjacent empty cells, creating a chain reaction until a numbered cell is reached.

Flags: Players can place flags on cells they believe contain mines. This helps them keep track of potential mine locations.

Winning the Game: The player wins when all non-mine cells are uncovered and flagged correctly.

Losing the Game: The game is lost if a player uncovers a cell containing a mine.

3] Strategies:

Use of Numbers: Analyze the numbers revealed on the board to deduce the locations of mines.

Flagging: Place flags on cells where you are confident there is a mine to avoid accidental clicks.

Logical Deduction: Use logical deduction to uncover safe cells and progress through the game.

Memorization: Memorize patterns and mines' locations to make informed decisions.

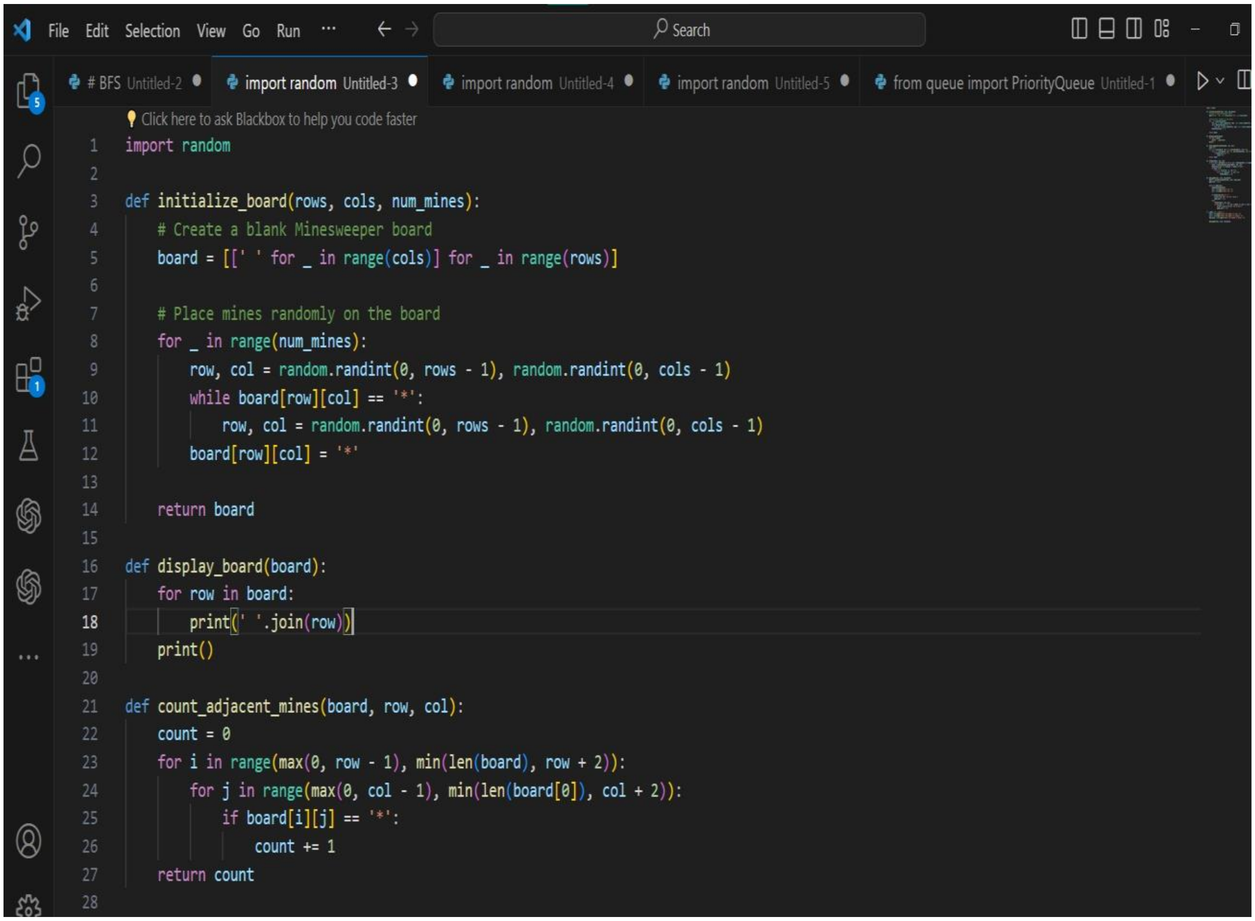
4] Python Implementation:

When implementing Minesweeper in Python, we will typically use a 2D list to represent the game board. We will need functions to handle uncovering cells, placing flags, checking for mines, and updating the board based on player actions.

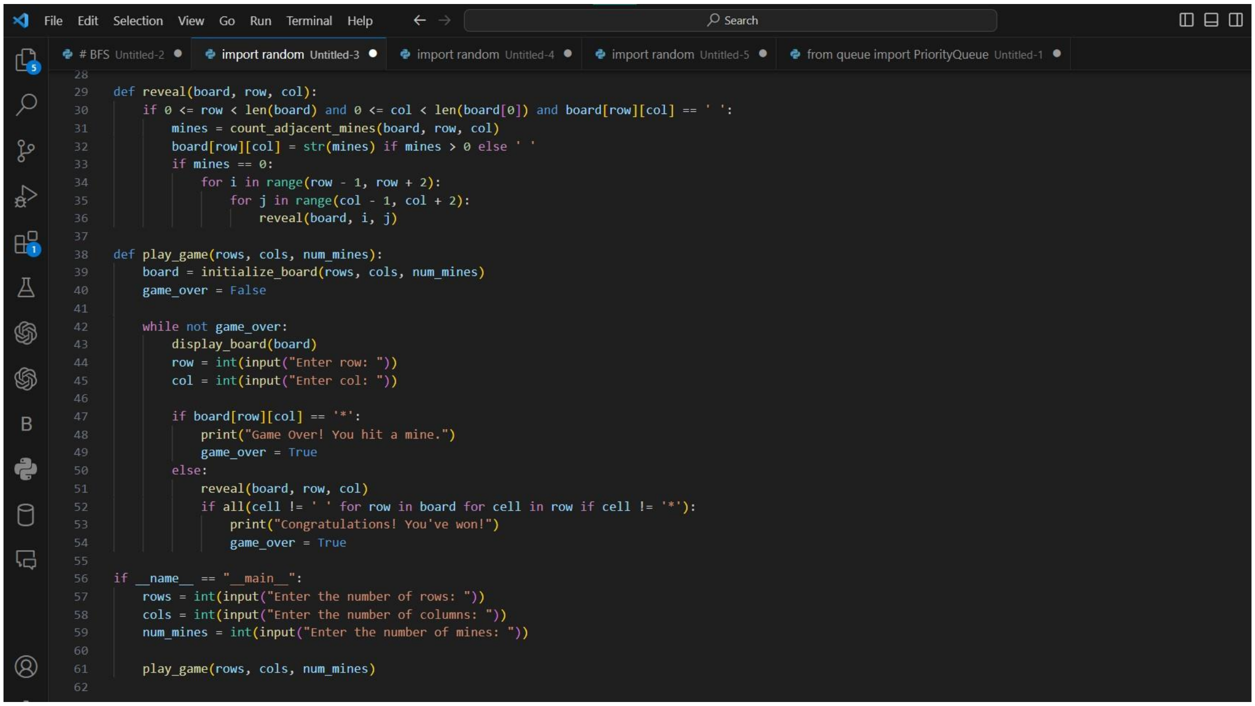
A simple Minesweeper implementation would involve managing the game state, handling user input, and updating the display after each move. You may use nested loops to iterate through the cells, and functions to reveal cells, place flags, and check for game over conditions.

Also we will handle edge cases, validate user input, and implement a game loop to keep the game running until the player wins or loses.

Python Code:

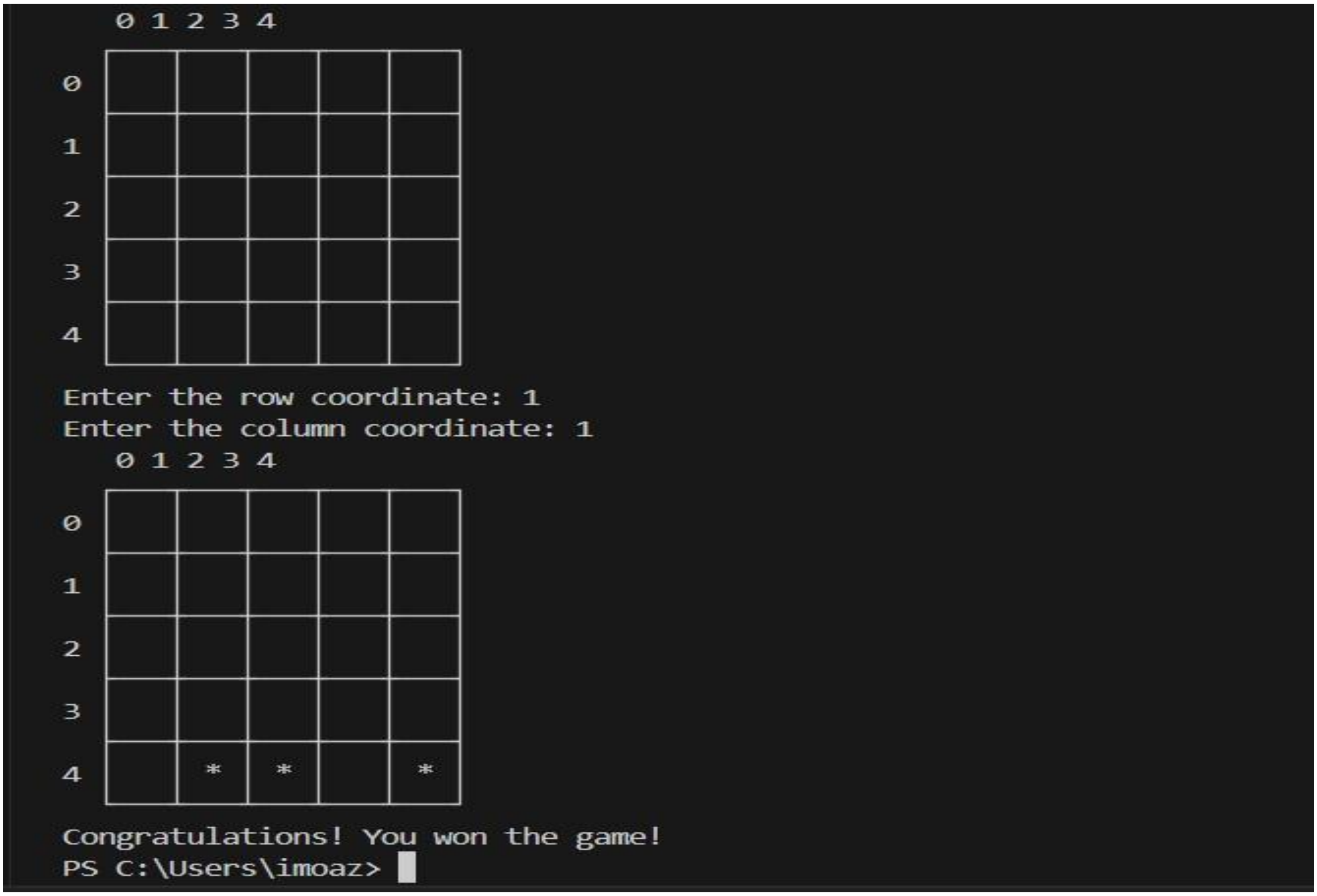

File Edit Selection View Go Run ···
Search
# BFS Untitled-2
import random Untitled-3
import random Untitled-4
import random Untitled-5
from queue import PriorityQueue Untitled-1
Click here to ask Blackbox to help you code faster

```python
import random

def initialize_board(rows, cols, num_mines):
    # Create a blank Minesweeper board
    board = [[' ' for _ in range(cols)] for _ in range(rows)]

    # Place mines randomly on the board
    for _ in range(num_mines):
        row, col = random.randint(0, rows - 1), random.randint(0, cols - 1)
        while board[row][col] == '*':
            row, col = random.randint(0, rows - 1), random.randint(0, cols - 1)
        board[row][col] = '*'

    return board

def display_board(board):
    for row in board:
        print(' '.join(row))
    print()

def count_adjacent_mines(board, row, col):
    count = 0
    for i in range(max(0, row - 1), min(len(board), row + 2)):
        for j in range(max(0, col - 1), min(len(board[0]), col + 2)):
            if board[i][j] == '*':
                count += 1
    return count

```

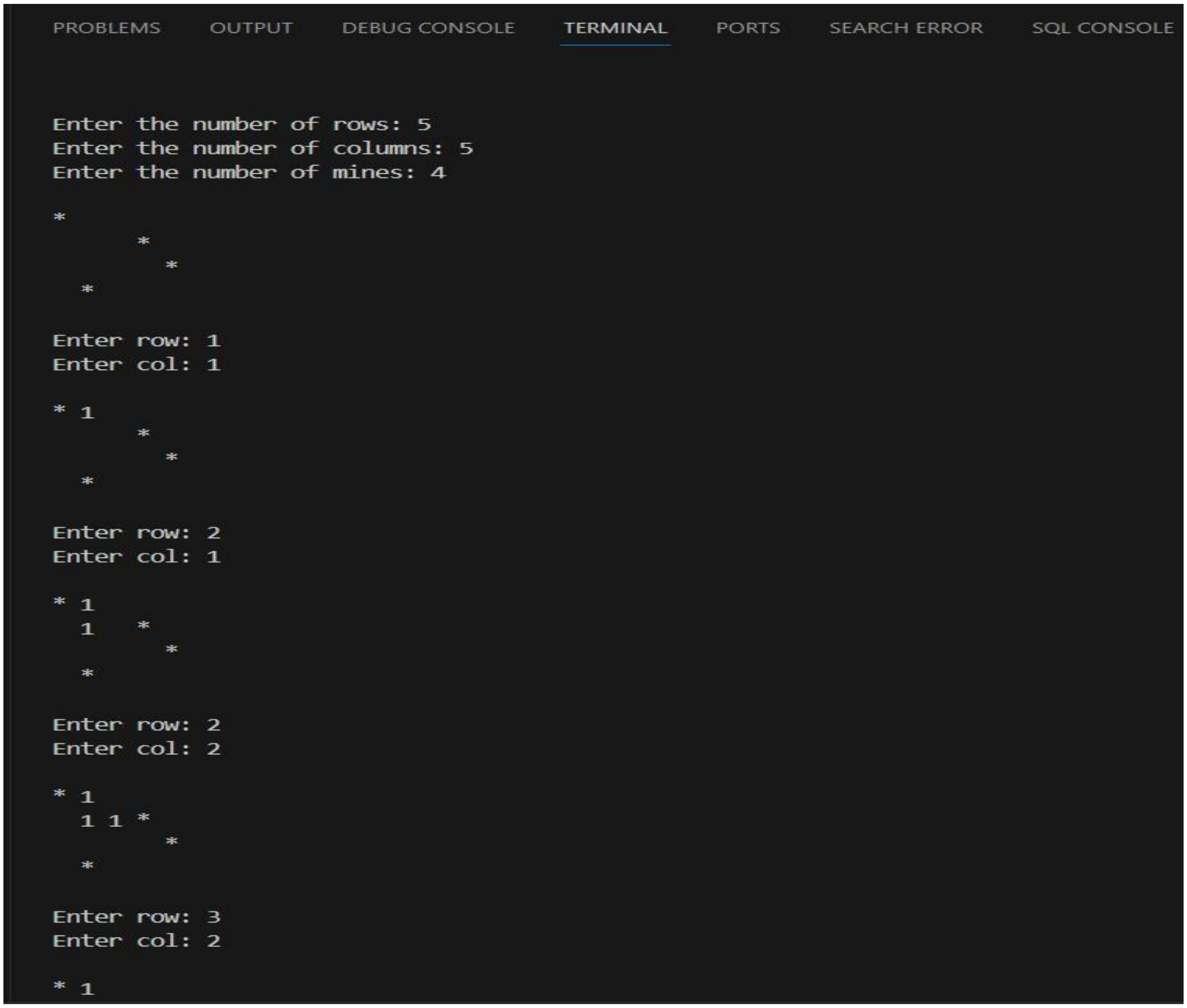


```python

def reveal(board, row, col):
    if 0 <= row < len(board) and 0 <= col < len(board[0]) and board[row][col] == ' ':
        mines = count_adjacent_mines(board, row, col)
        board[row][col] = str(mines) if mines > 0 else ' '
        if mines == 0:
            for i in range(row - 1, row + 2):
                for j in range(col - 1, col + 2):
                    reveal(board, i, j)

def play_game(rows, cols, num_mines):
    board = initialize_board(rows, cols, num_mines)
    game_over = False

    while not game_over:
        display_board(board)
        row = int(input("Enter row: "))
        col = int(input("Enter col: "))

        if board[row][col] == '*':
            print("Game Over! You hit a mine.")
            game_over = True
        else:
            reveal(board, row, col)
            if all(cell != ' ' for row in board for cell in row if cell != '*'):
                print("Congratulations! You've won!")
                game_over = True

if __name__ == "__main__":
    rows = int(input("Enter the number of rows: "))
    cols = int(input("Enter the number of columns: "))
    num_mines = int(input("Enter the number of mines: "))

    play_game(rows, cols, num_mines)

```

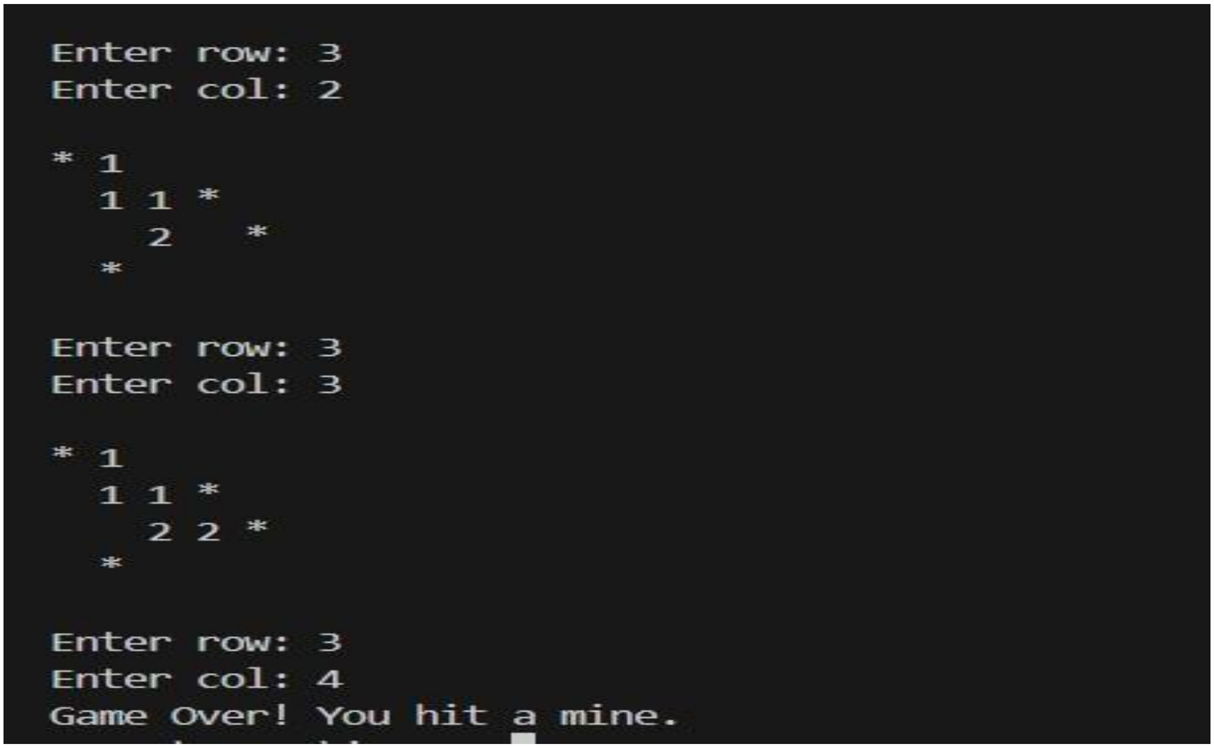
Some different outputs:



```
    0 1 2 3 4
   ┌───┬───┬───┬───┬───┐
 0 │   │   │   │   │   │
   ├───┼───┼───┼───┼───┤
 1 │   │   │   │   │   │
   ├───┼───┼───┼───┼───┤
 2 │   │   │   │   │   │
   ├───┼───┼───┼───┼───┤
 3 │   │   │   │   │   │
   ├───┼───┼───┼───┼───┤
 4 │   │   │   │   │   │
   └───┴───┴───┴───┴───┘
Enter the row coordinate: 1
Enter the column coordinate: 1
    0 1 2 3 4
   ┌───┬───┬───┬───┬───┐
 0 │   │   │   │   │   │
   ├───┼───┼───┼───┼───┤
 1 │   │   │   │   │   │
   ├───┼───┼───┼───┼───┤
 2 │   │   │   │   │   │
   ├───┼───┼───┼───┼───┤
 3 │   │   │   │   │   │
   ├───┼───┼───┼───┼───┤
 4 │   │ * │ * │   │ * │
   └───┴───┴───┴───┴───┘
Congratulations! You won the game!
PS C:\Users\imoaz>
```

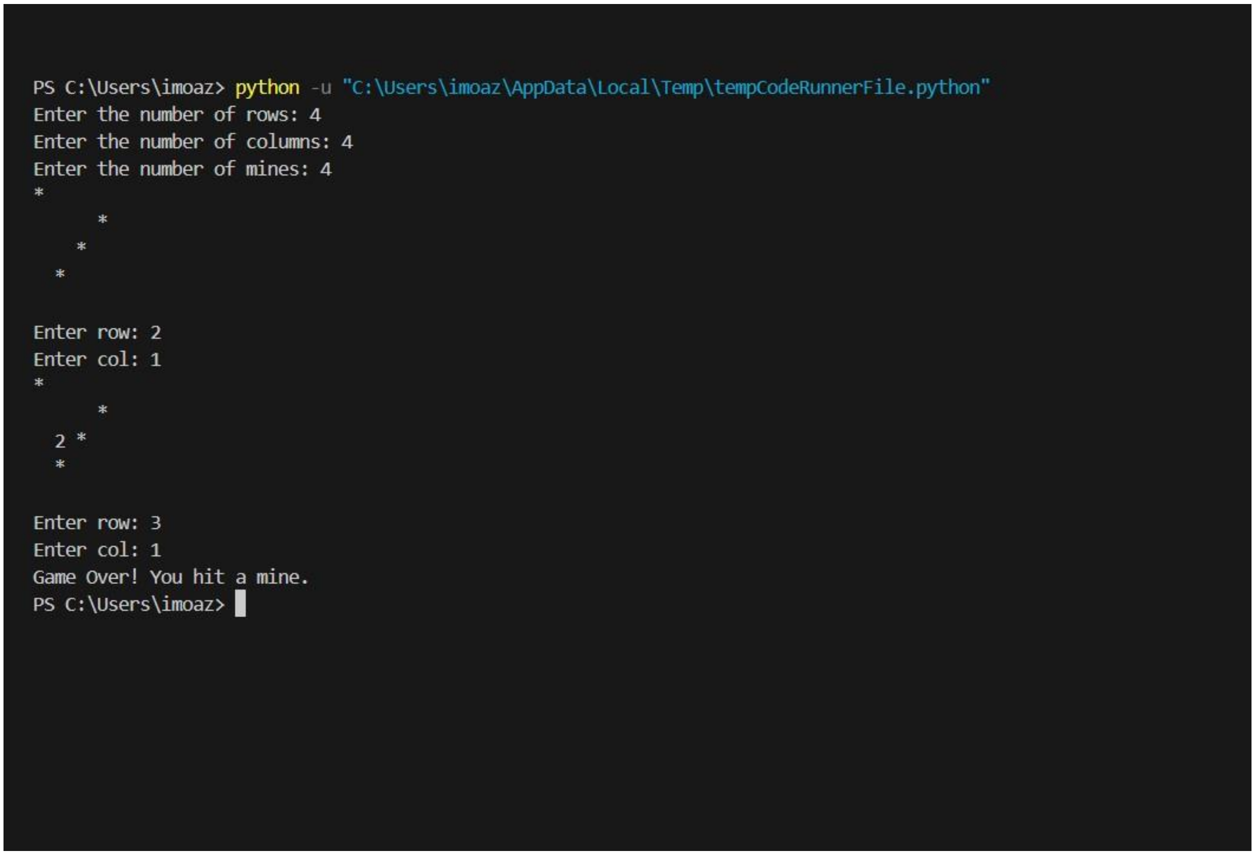


```
Enter the number of rows: 5
Enter the number of columns: 5
Enter the number of mines: 4

*
      *
        *
  *

Enter row: 1
Enter col: 1

* 1
      *
        *
  *

Enter row: 2
Enter col: 1

* 1
  1   *
        *
  *

Enter row: 2
Enter col: 2

* 1
  1 1 *
        *
  *

Enter row: 3
Enter col: 2

* 1
```



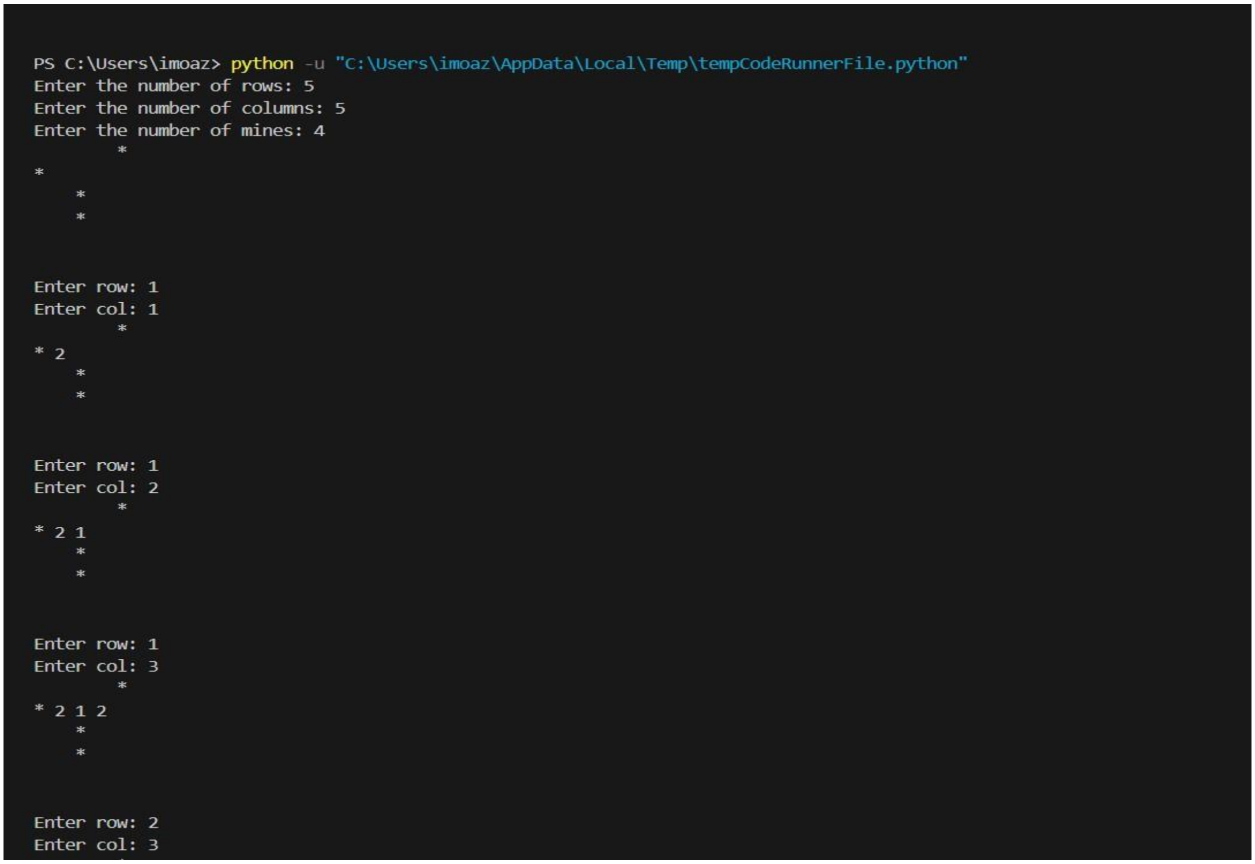
```
Enter row: 3
Enter col: 2

* 1
  1 1 *
    2   *
  *

Enter row: 3
Enter col: 3

* 1
  1 1 *
    2 2 *
  *

Enter row: 3
Enter col: 4
Game Over! You hit a mine.
```

****

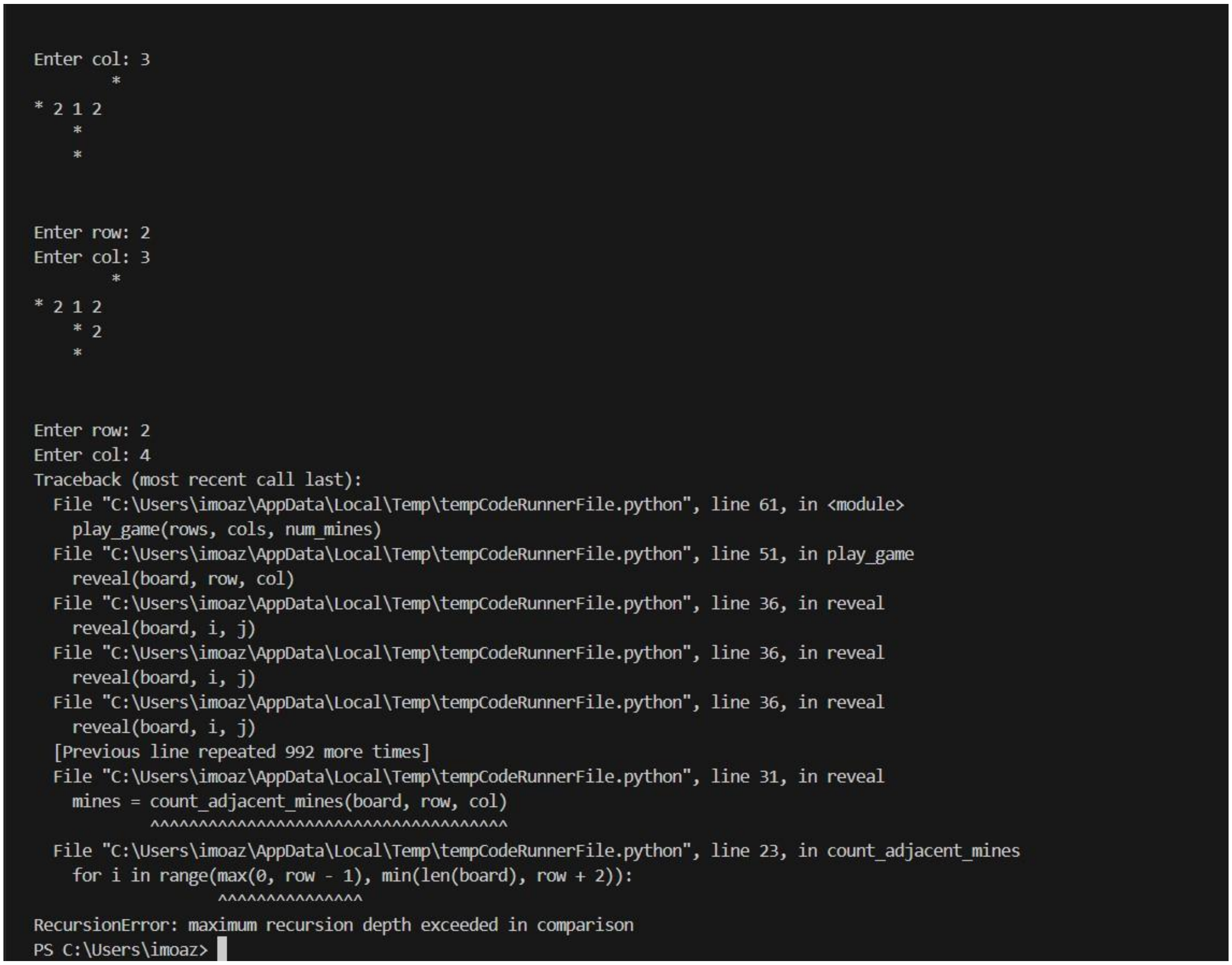


```
PS C:\Users\imoaz> python -u "C:\Users\imoaz\AppData\Local\Temp\tempCodeRunnerFile.python"
Enter the number of rows: 4
Enter the number of columns: 4
Enter the number of mines: 4
*
      *
    *
  *

Enter row: 2
Enter col: 1
*
      *
  2 *
  *

Enter row: 3
Enter col: 1
Game Over! You hit a mine.
PS C:\Users\imoaz>
```

****



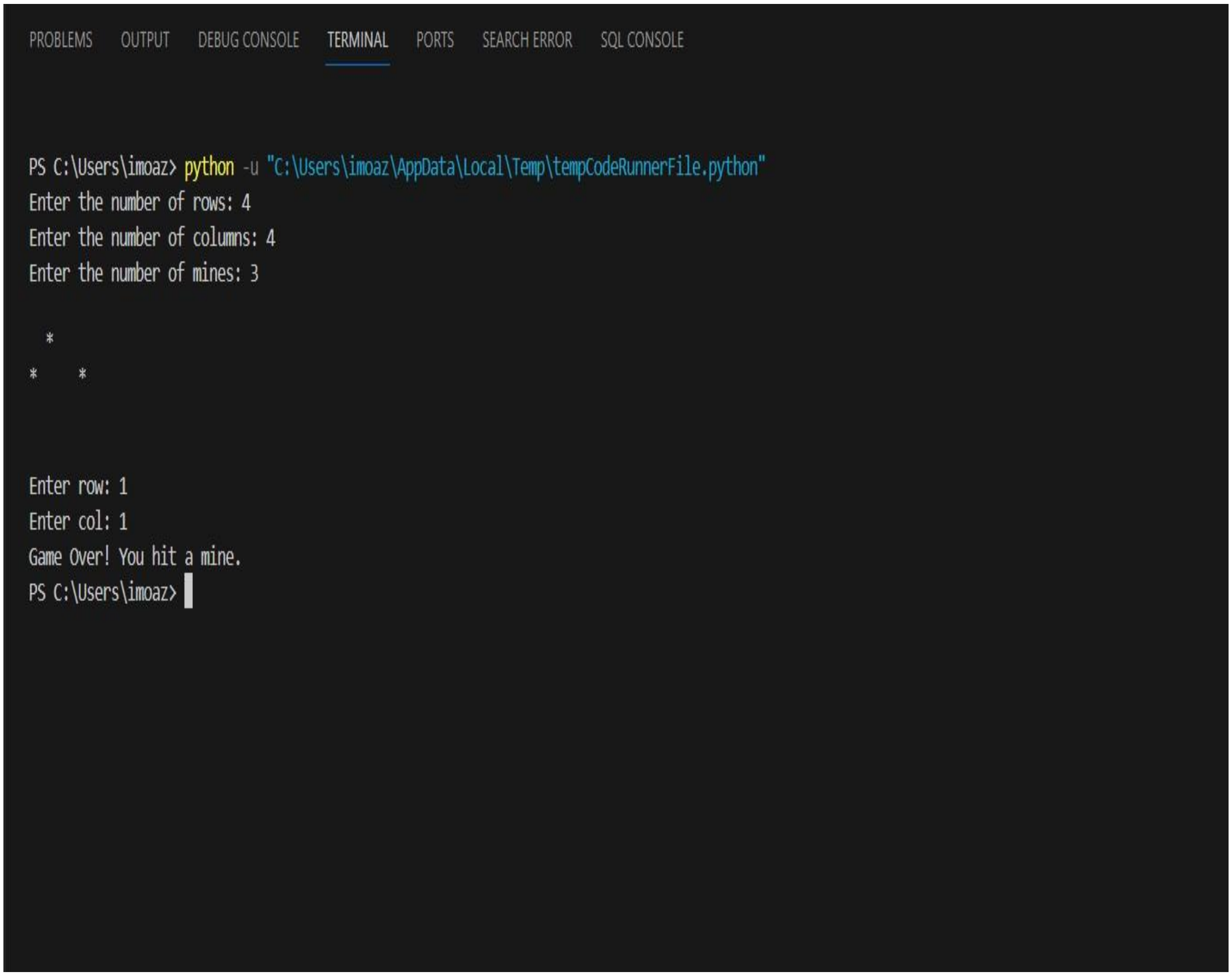
```
PS C:\Users\imoaz> python -u "C:\Users\imoaz\AppData\Local\Temp\tempCodeRunnerFile.python"
Enter the number of rows: 5
Enter the number of columns: 5
Enter the number of mines: 4
        *
*
    *
    *


Enter row: 1
Enter col: 1
        *
* 2
    *
    *


Enter row: 1
Enter col: 2
        *
* 2 1
    *
    *


Enter row: 1
Enter col: 3
        *
* 2 1 2
    *
    *


Enter row: 2
Enter col: 3
```



```
Enter col: 3
        *
* 2 1 2
    *
    *


Enter row: 2
Enter col: 3
        *
* 2 1 2
    * 2
    *


Enter row: 2
Enter col: 4
Traceback (most recent call last):
  File "C:\Users\imoaz\AppData\Local\Temp\tempCodeRunnerFile.python", line 61, in <module>
    play_game(rows, cols, num_mines)
  File "C:\Users\imoaz\AppData\Local\Temp\tempCodeRunnerFile.python", line 51, in play_game
    reveal(board, row, col)
  File "C:\Users\imoaz\AppData\Local\Temp\tempCodeRunnerFile.python", line 36, in reveal
    reveal(board, i, j)
  File "C:\Users\imoaz\AppData\Local\Temp\tempCodeRunnerFile.python", line 36, in reveal
    reveal(board, i, j)
  File "C:\Users\imoaz\AppData\Local\Temp\tempCodeRunnerFile.python", line 36, in reveal
    reveal(board, i, j)
  [Previous line repeated 992 more times]
  File "C:\Users\imoaz\AppData\Local\Temp\tempCodeRunnerFile.python", line 31, in reveal
    mines = count_adjacent_mines(board, row, col)
            ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\imoaz\AppData\Local\Temp\tempCodeRunnerFile.python", line 23, in count_adjacent_mines
    for i in range(max(0, row - 1), min(len(board), row + 2)):
                   ^^^^^^^^^^^^^^^
RecursionError: maximum recursion depth exceeded in comparison
PS C:\Users\imoaz>
```

****

****



```
PS C:\Users\imoaz> python -u "C:\Users\imoaz\AppData\Local\Temp\tempCodeRunnerFile.python"
Enter the number of rows: 4
Enter the number of columns: 4
Enter the number of mines: 3

  *
*     *



Enter row: 1
Enter col: 1
Game Over! You hit a mine.
PS C:\Users\imoaz>
```