**MINESWEEPER GAME**

**Name : Veeransh Shah**

**Reg ID: 221070063**

**Python Mini Project**

**Aim:**

The main aim of this was to build an Interactive Minesweeper Game with GUI, and get used to more such Python libraries and other frameworks like tkinter, numpy, random, re

**Theory:**

1. **Board Class:**
   * **\_\_init\_\_(self, dim\_size, num\_bombs)**: Initializes a Minesweeper board with the specified dimensions (**dim\_size** x **dim\_size**) and number of bombs (**num\_bombs**).
   * **make\_new\_board(self)**: Randomly places bombs on the board and returns the initial board state.
   * **assign\_values\_to\_board(self)**: Assigns values to non-bomb cells indicating the number of neighboring bombs.
   * **get\_num\_neighboring\_bombs(self, row, col)**: Counts the number of neighboring bombs for a given cell.
   * **dig(self, row, col)**: Reveals a cell on the board. If the revealed cell contains a bomb, the game is lost. If the revealed cell has neighboring bombs, it shows the count. If the revealed cell is empty, it recursively reveals neighboring cells.
   * **\_\_str\_\_(self)**: Generates a string representation of the current state of the board.
2. **play Function:**
   * **play(dim\_size=10, num\_bombs=10)**: Sets up and plays the Minesweeper game.
   * Creates an instance of the **Board** class with the specified dimensions and number of bombs.
   * Enters a loop where the player is prompted to input the coordinates of the cell they want to dig.
   * The game continues until either all non-bomb cells are revealed or a bomb is dug.
   * If the game is won, a victory message is displayed; otherwise, a game over message is shown.
3. **User Input Handling:**
   * The player provides input in the format "row, col" to specify the cell they want to dig.
   * Input is processed using regular expressions to extract the row and column values.
4. **Printing the Board:**
   * The current state of the Minesweeper board is printed after each move, showing the revealed cells and their values.
5. **Game Outcome:**
   * If the player successfully reveals all non-bomb cells, a victory message is displayed.
   * If the player digs a cell containing a bomb, a game over message is shown along with the revealed board.

**Code:**

import random

import re

class Board:

    def \_\_init\_\_(self, dim\_size, num\_bombs):

        self.dim\_size = dim\_size

        self.num\_bombs = num\_bombs

        self.board = self.make\_new\_board()

        self.assign\_values\_to\_board()

        self.dug = set()

    def make\_new\_board(self):

        board = [[None for \_ in range(self.dim\_size)] for \_ in range(self.dim\_size)]

        bombs\_planted = 0

        while bombs\_planted < self.num\_bombs:

            loc = random.randint(0, self.dim\_size\*\*2 - 1)

            row = loc // self.dim\_size

            col = loc % self.dim\_size

            if board[row][col] == '\*':

                continue

            board[row][col] = '\*'

            bombs\_planted += 1

        return board

    def assign\_values\_to\_board(self):

        for r in range(self.dim\_size):

            for c in range(self.dim\_size):

                if self.board[r][c] == '\*':

                    continue

                self.board[r][c] = self.get\_num\_neighboring\_bombs(r, c)

    def get\_num\_neighboring\_bombs(self, row, col):

        num\_neighboring\_bombs = 0

        for r in range(max(0, row-1), min(self.dim\_size-1, row+1)+1):

            for c in range(max(0, col-1), min(self.dim\_size-1, col+1)+1):

                if r == row and c == col:

                    continue

                if self.board[r][c] == '\*':

                    num\_neighboring\_bombs += 1

        return num\_neighboring\_bombs

    def dig(self, row, col):

        self.dug.add((row, col))

        if self.board[row][col] == '\*':

            return False

        elif self.board[row][col] > 0:

            return True

        for r in range(max(0, row-1), min(self.dim\_size-1, row+1)+1):

            for c in range(max(0, col-1), min(self.dim\_size-1, col+1)+1):

                if (r, c) in self.dug:

                    continue

                self.dig(r, c)

        return True

    def \_\_str\_\_(self):

        visible\_board = [[None for \_ in range(self.dim\_size)] for \_ in range(self.dim\_size)]

        for row in range(self.dim\_size):

            for col in range(self.dim\_size):

                if (row,col) in self.dug:

                    visible\_board[row][col] = str(self.board[row][col])

                else:

                    visible\_board[row][col] = ' '

        string\_rep = ''

        widths = []

        for idx in range(self.dim\_size):

            columns = map(lambda x: x[idx], visible\_board)

            widths.append(

                len(

                    max(columns, key = len)

                )

            )

        # print the csv strings

        indices = [i for i in range(self.dim\_size)]

        indices\_row = '   '

        cells = []

        for idx, col in enumerate(indices):

            format = '%-' + str(widths[idx]) + "s"

            cells.append(format % (col))

        indices\_row += '  '.join(cells)

        indices\_row += '  \n'

        for i in range(len(visible\_board)):

            row = visible\_board[i]

            string\_rep += f'{i} |'

            cells = []

            for idx, col in enumerate(row):

                format = '%-' + str(widths[idx]) + "s"

                cells.append(format % (col))

            string\_rep += ' |'.join(cells)

            string\_rep += ' |\n'

        str\_len = int(len(string\_rep) / self.dim\_size)

        string\_rep = indices\_row + '-'\*str\_len + '\n' + string\_rep + '-'\*str\_len

        return string\_rep

def play(dim\_size=10, num\_bombs=10):

    board = Board(dim\_size, num\_bombs)

    safe = True

    while len(board.dug) < board.dim\_size \*\* 2 - num\_bombs:

        print(board)

        user\_input = re.split(',(\\s)\*', input("Where would you like to dig? Input as row,col: "))  # '0, 3'

        row, col = int(user\_input[0]), int(user\_input[-1])

        if row < 0 or row >= board.dim\_size or col < 0 or col >= dim\_size:

            print("Invalid location. Try again.")

            continue

        safe = board.dig(row, col)

        if not safe:

            break

    if safe:

        print("CONGRATULATIONS!!!! YOU ARE VICTORIOUS!")

    else:

        print("GAME OVER!!! BETTER LUCK NEXT TIME :(")

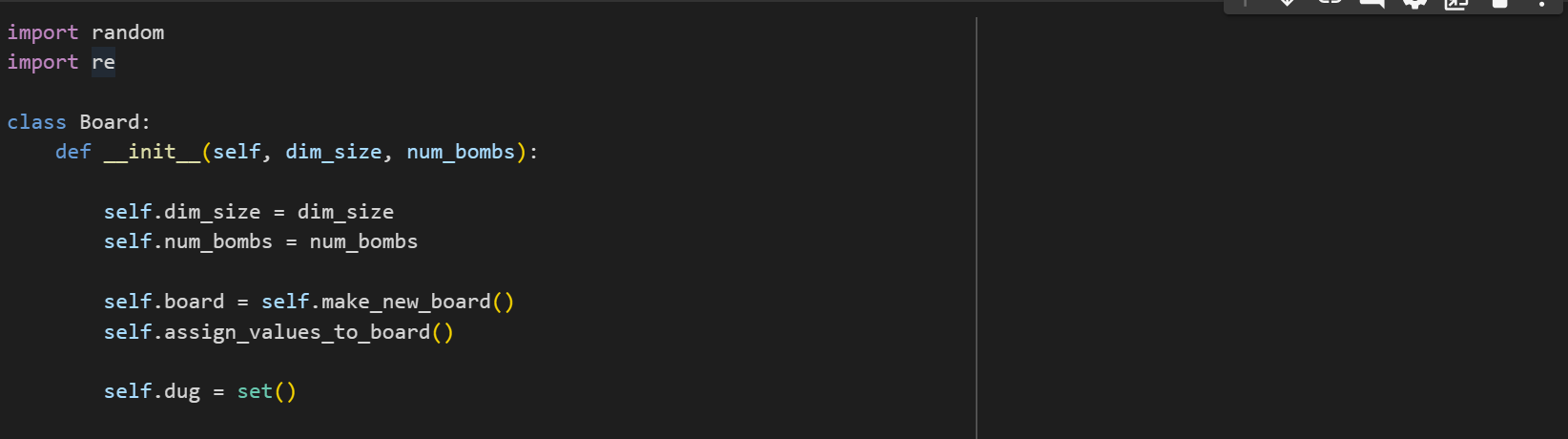
        board.dug = [(r,c) for r in range(board.dim\_size) for c in range(board.dim\_size)]

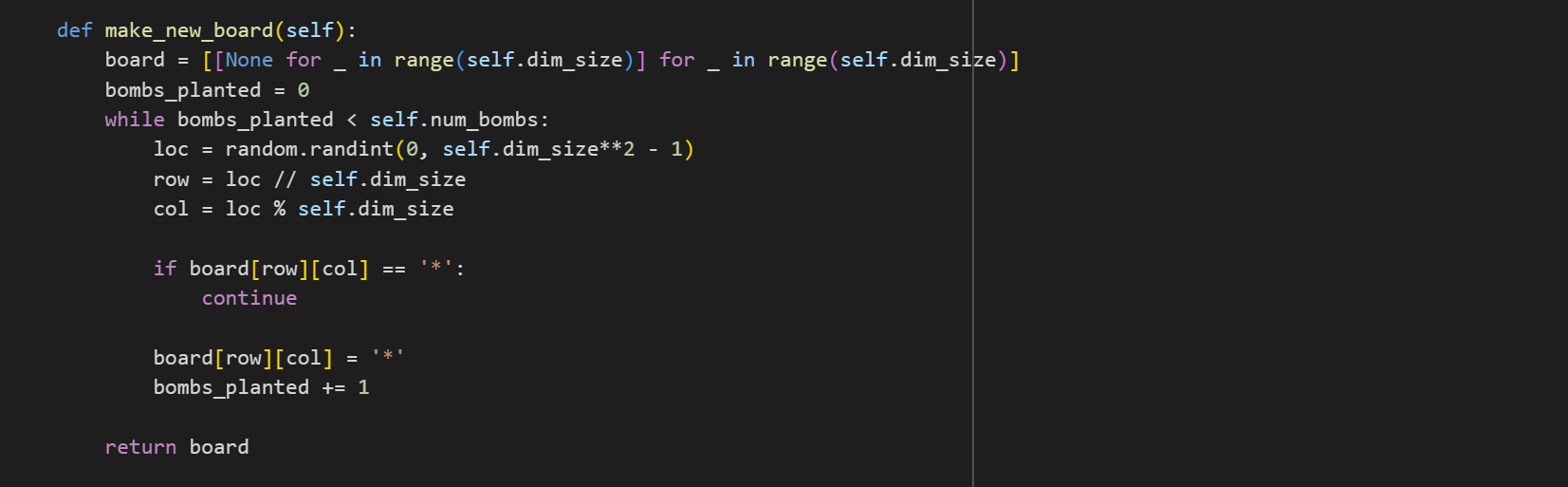
        print(board)

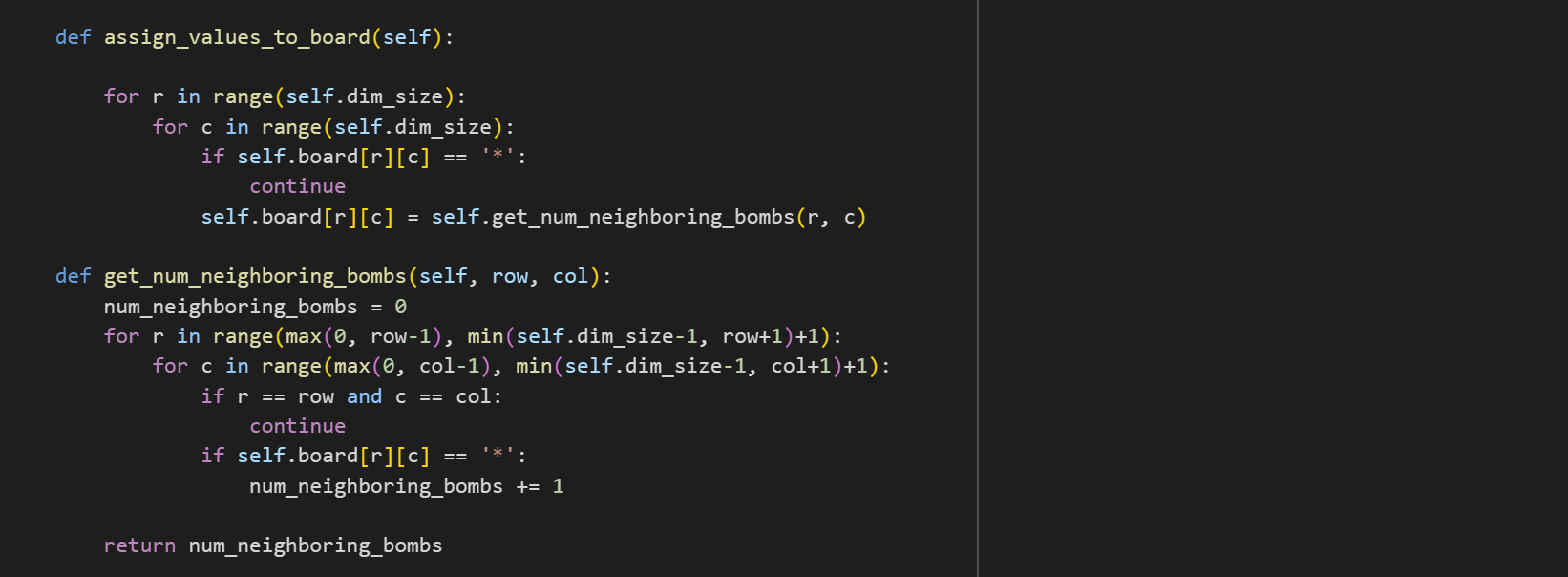
if \_\_name\_\_ == '\_\_main\_\_':

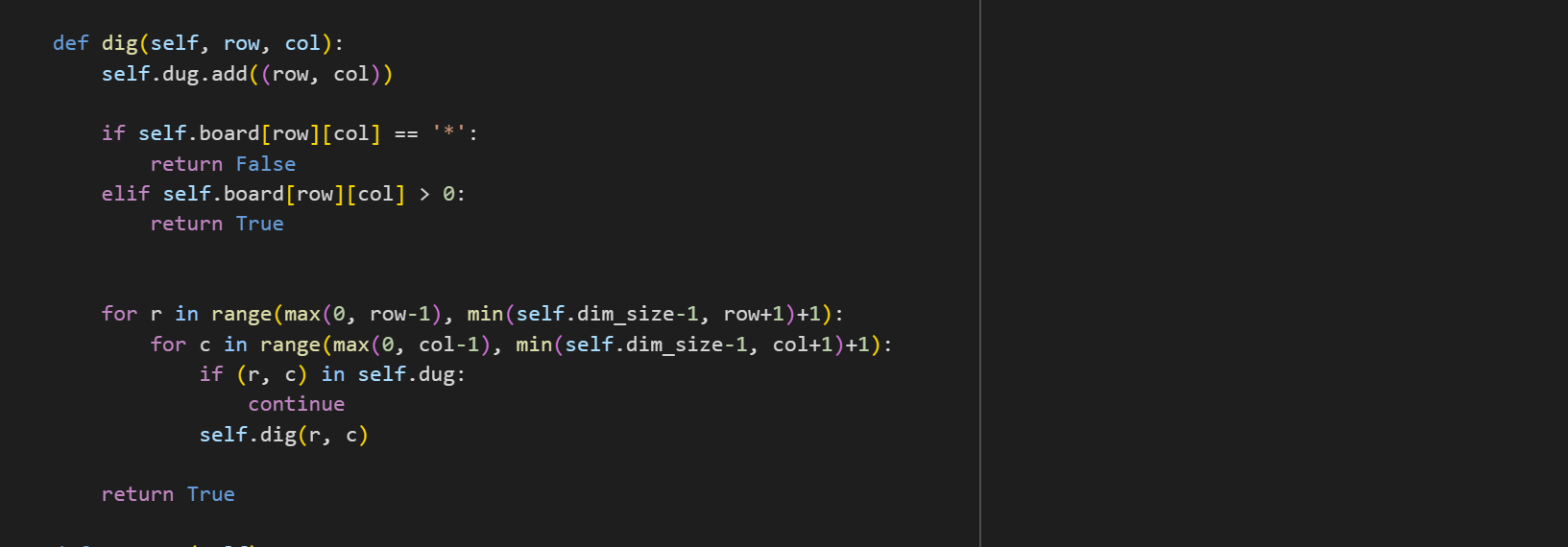
    play()

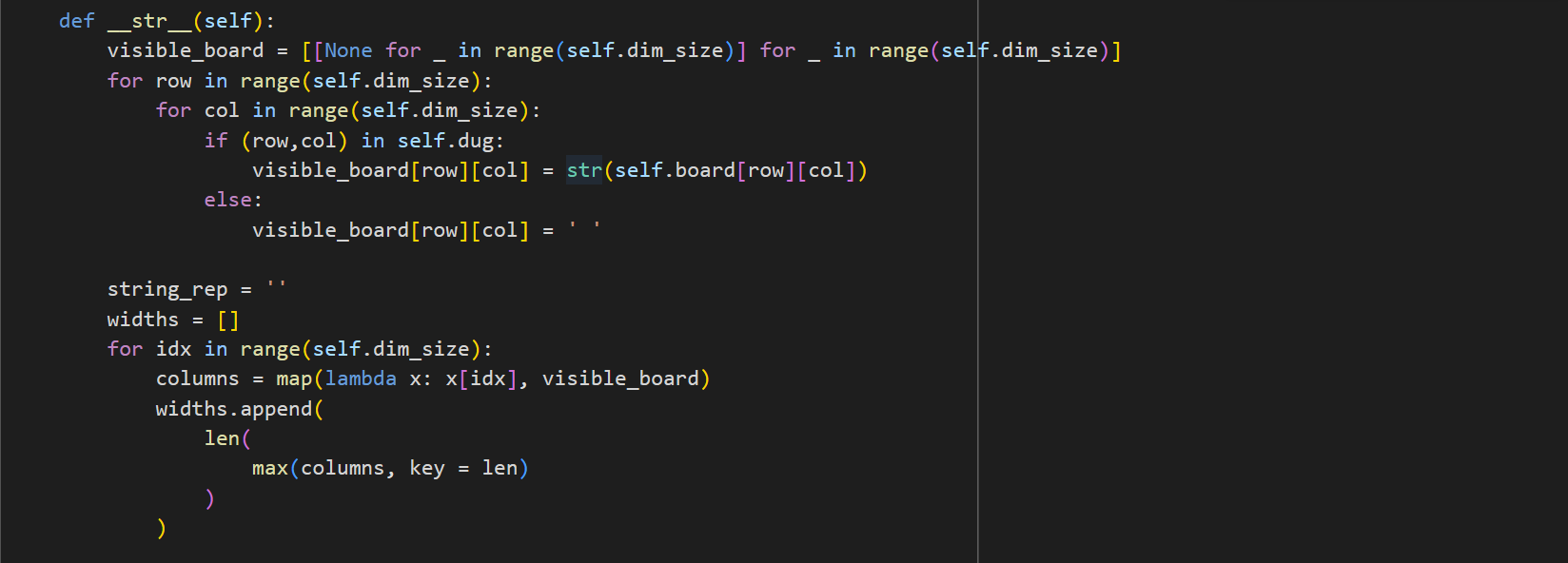
**Code Screen Shots:**

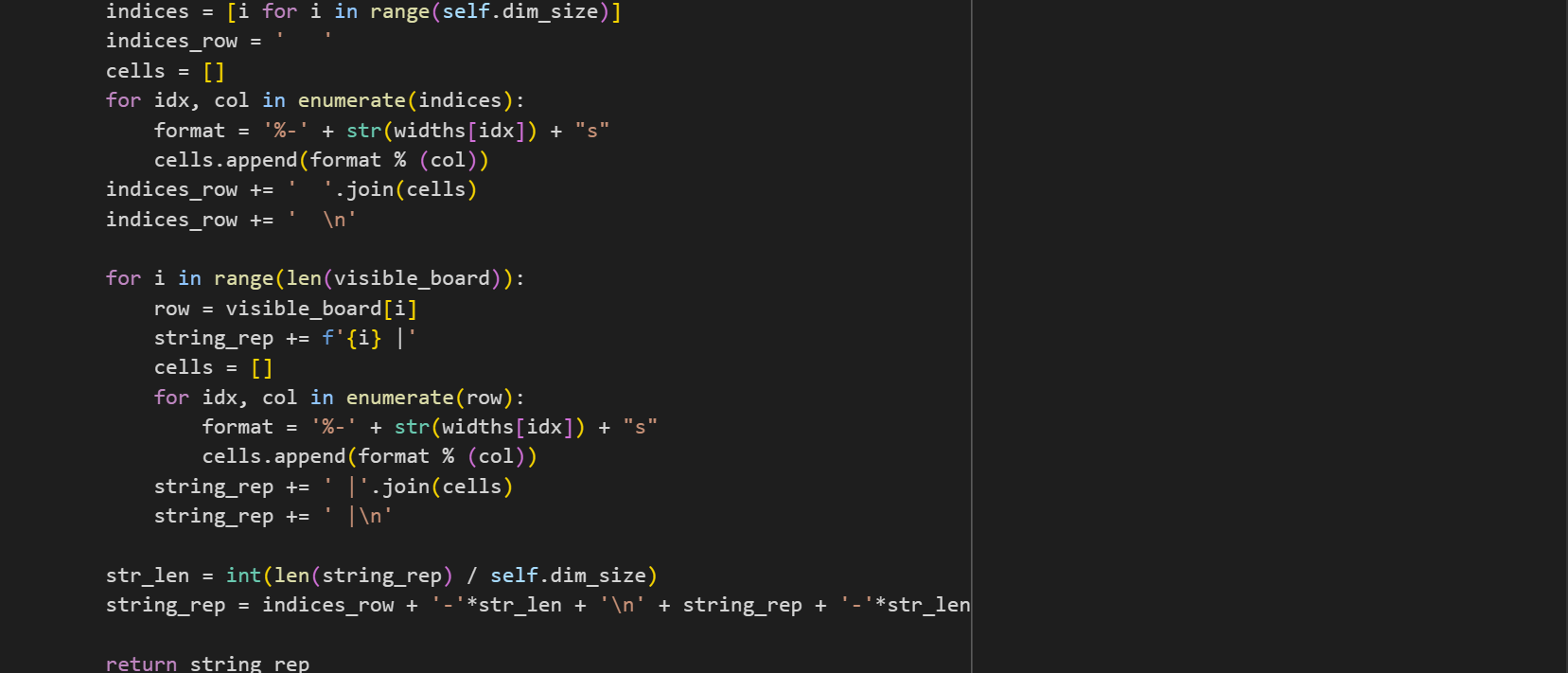


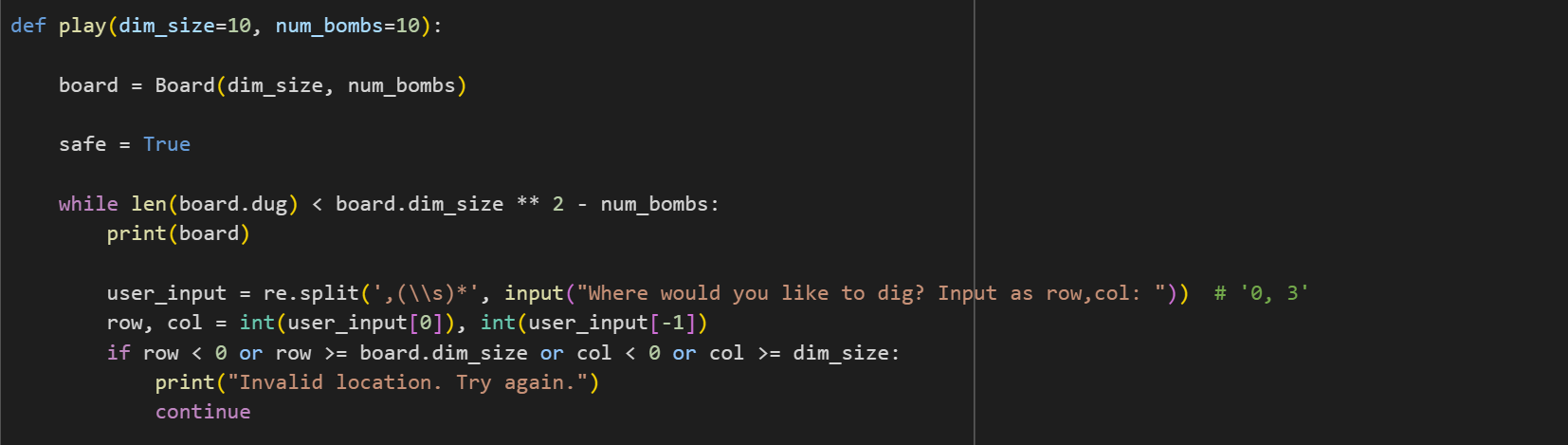


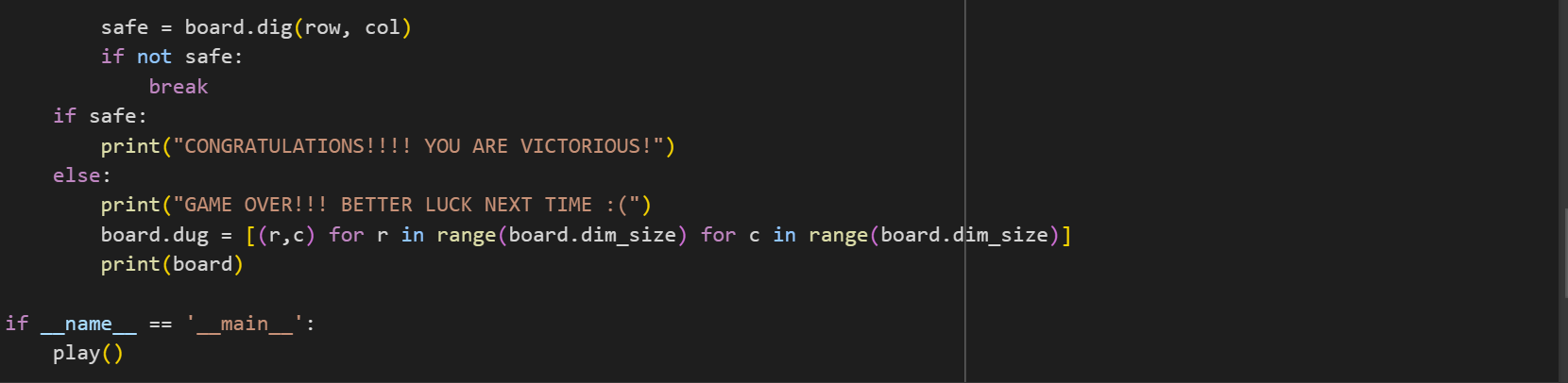






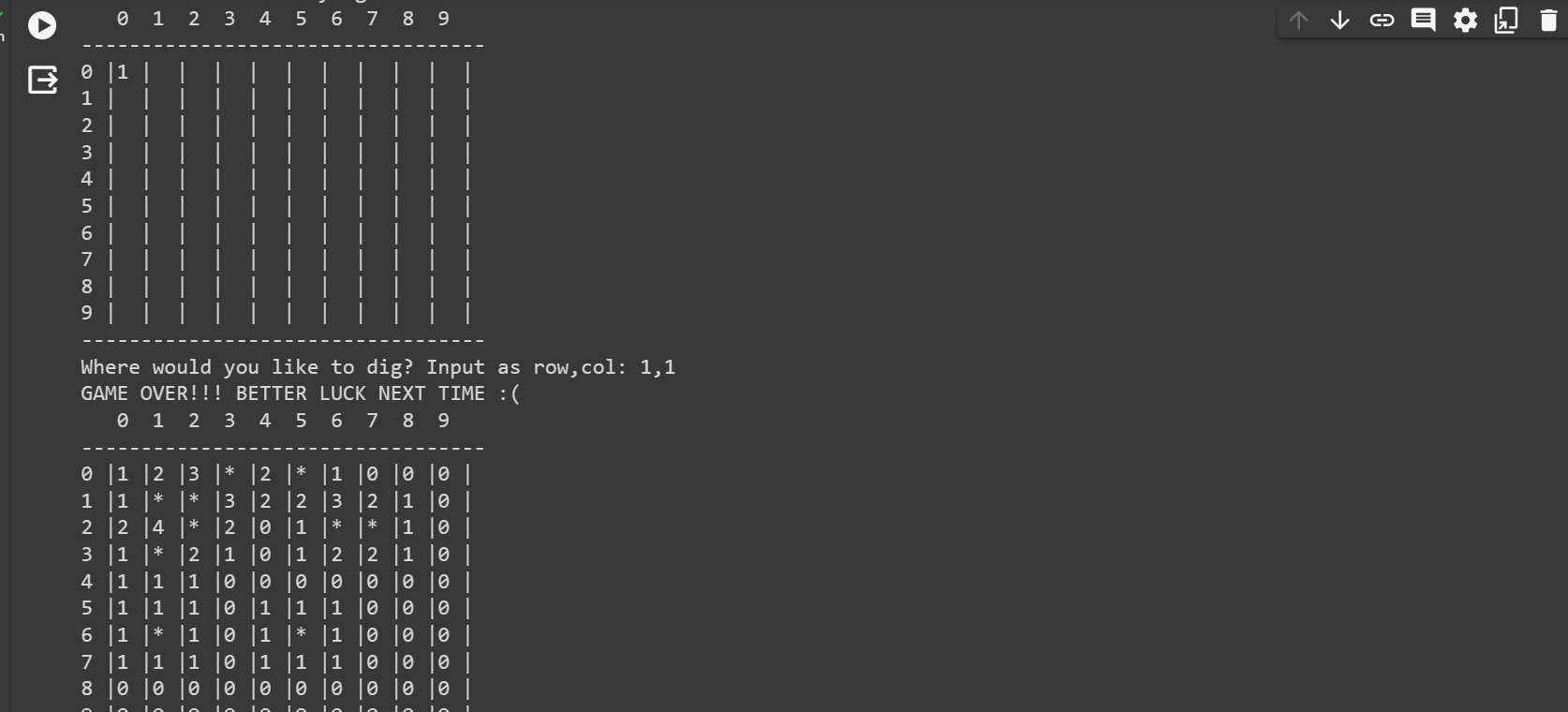






**Output Screen Shot:**





**Conclusion:**

The Minesweeper mini-project has been successfully developed into an interactive and engaging game with a graphical user interface (GUI). The core Minesweeper mechanics, including bomb placement, neighbouring cell counts, and cell revealing logic, have been faithfully implemented.

In summary, the Minesweeper mini-project has been successfully transformed into a feature-rich, interactive game.