

2D Game Board with Movement and Event Triggers (Minesweeper Game)

Submitted By: Muneeb Ahmad

Registration No.: 12311550

Acknowledgment I would like to express my sincere gratitude to my respected faculty member for guiding me in the successful completion of this project titled "2D Game Board with Movement and Event Triggers (Minesweeper)". I also thank my peers and family for their continuous support.

Table of Contents

1. Introduction
2. Objective
3. Technologies Used
4. Game Overview
5. Features and Flow
6. Full Code (HTML/CSS/JavaScript)
7. Full Code (Python)
8. Screenshot (HTML/CSS/JavaScript)
9. Screenshot (Python)
10. Conclusion
11. Introduction This project showcases a simple and interactive 2D game called Minesweeper, implemented using HTML, CSS, and JavaScript for a web-based version, and Python for a console-based version. The game involves user interaction with a dynamic board that responds to mouse events (in the HTML/CSS/JavaScript version) and text-based input (in the Python version). It demonstrates the application of grid-based layout, recursive logic, and DOM manipulation (for web) or console output (for Python) to simulate a classic game.
12. Objective To design and implement a 2D game board that includes:
 - Object-based movement logic
 - Event-driven interaction
 - A responsive and interactive user interface (for web) or clear console interface (for Python)
3. Technologies Used HTML - Page structure and grid layout for the web version CSS - Styling game elements and layout for the web version JavaScript - Game logic and interactivity for the web version Python - Game logic and console interactivity for the text-based version
4. Game Overview
 - Grid Size: 8 x 8
 - Mines: 10 random positions
 - User clicks to reveal cells (web version) or enters coordinates (console version)

- Flagging option for suspected mines
 - Loss on clicking a mine, win on clearing all safe cells
5. Features and Flow
- 2D Grid Setup: Board is dynamically generated using nested arrays and grid layout (web) or nested lists (Python).
 - Event Triggers:
 - Left-click: Reveals a cell (web)
 - Right-click: Flags a cell (web)
 - Text input: Reveals or flags a cell (Python)
 - Game ends on mine click
 - Movement: Logical movement via recursive reveal of surrounding cells with zero mines.
 - Scoring: No score count but victory is detected by revealing all safe cells.
6. Full Code (HTML/CSS/JavaScript) The complete HTML, CSS, and JavaScript code for the Minesweeper game is provided below:

```
<!DOCTYPE html>
```

```
<html lang="en">
```

```
<head>
```

```
<meta charset="UTF-8">
```

```
<title>Minesweeper Game</title>
```

```
<style>
```

```
  body {
```

```
    font-family: Arial;
```

```
    text-align: center;
```

```
    background: #f2f2f2;
```

```
  }
```

```
  #game-board {
```

```
    display: grid;
```

```
    grid-template-columns: repeat(8, 40px);
```

```
    margin: 20px auto;
```

```
    gap: 2px;
```

```
    width: fit-content;
```

```
  }
```

```
.cell {
  width: 40px;
  height: 40px;
  background: #c4c4c4;
  border: 1px solid #888;
  font-size: 18px;
  font-weight: bold;
  cursor: pointer;
  user-select: none;
}

.cell.revealed {
  background: #e0e0e0;
  cursor: default;
}

.cell.mine {
  background: #d9534f;
  color: white;
}

.cell.flagged {
  background: #5bc0de;
  color: white;
}

#status {
  font-size: 18px;
  margin-top: 10px;
}

</style>
</head>
<body>

<h2>Minesweeper</h2>

<div id="game-board"></div>
```

```
<p id="status"></p>
```

```
<script>
```

```
    const board = document.getElementById("game-board");
```

```
    const status = document.getElementById("status");
```

```
    let rows = 8;
```

```
    let cols = 8;
```

```
    let mineCount = 10;
```

```
    let cells = [];
```

```
function createBoard() {
```

```
    cells = [];
```

```
    board.innerHTML = "";
```

```
    for (let r = 0; r < rows; r++) {
```

```
        cells[r] = [];
```

```
        for (let c = 0; c < cols; c++) {
```

```
            let cell = document.createElement("div");
```

```
            cell.classList.add("cell");
```

```
            cell.dataset.row = r;
```

```
            cell.dataset.col = c;
```

```
            board.appendChild(cell);
```

```
            cells[r][c] = { element: cell, mine: false, revealed: false, flagged: false, count: 0 };
```

```
        }
```

```
    }
```

```
    placeMines();
```

```
    countNumbers();
```

```
    addClickEvents();
```

```
}
```

```
function placeMines() {
```

```
    let placed = 0;
```

```
    while (placed < mineCount) {
```

```

let r = Math.floor(Math.random() * rows);
let c = Math.floor(Math.random() * cols);
if (!cells[r][c].mine) {
    cells[r][c].mine = true;
    placed++;
}
}
}

```

```

function countNumbers() {
    for (let r = 0; r < rows; r++) {
        for (let c = 0; c < cols; c++) {
            if (cells[r][c].mine) continue;
            let count = 0;
            for (let i = -1; i <= 1; i++) {
                for (let j = -1; j <= 1; j++) {
                    let nr = r + i;
                    let nc = c + j;
                    if (nr >= 0 && nr < rows && nc >= 0 && nc < cols && cells[nr][nc].mine) {
                        count++;
                    }
                }
            }
            cells[r][c].count = count;
        }
    }
}

```

```

function addClickEvents() {
    board.addEventListener("contextmenu", (e) => e.preventDefault());
    board.querySelectorAll(".cell").forEach(cell => {

```

```

cell.addEventListener("click", function () {
    let r = +this.dataset.row;
    let c = +this.dataset.col;
    reveal(r, c);
});
cell.addEventListener("contextmenu", function () {
    let r = +this.dataset.row;
    let c = +this.dataset.col;
    flag(r, c);
});
});
}

```

```

function reveal(r, c) {
    let cell = cells[r][c];
    if (cell.revealed || cell.flagged) return;
    cell.revealed = true;
    cell.element.classList.add("revealed");
    if (cell.mine) {
        cell.element.classList.add("mine");
        cell.element.innerText = "💣";
        status.innerText = "💣 Game Over!";
        revealAllMines();
        disableBoard();
        return;
    }
    if (cell.count > 0) {
        cell.element.innerText = cell.count;
    } else {
        for (let i = -1; i <= 1; i++) {
            for (let j = -1; j <= 1; j++) {

```

```

        let nr = r + i;
        let nc = c + j;
        if (nr >= 0 && nr < rows && nc >= 0 && nc < cols) {
            reveal(nr, nc);
        }
    }
}
}
checkWin();
}

```

```

function flag(r, c) {
    let cell = cells[r][c];
    if (cell.revealed) return;
    cell.flagged = !cell.flagged;
    cell.element.classList.toggle("flagged");
    cell.element.innerText = cell.flagged ? "🚩" : "";
}

```

```

function revealAllMines() {
    for (let row of cells) {
        for (let cell of row) {
            if (cell.mine) {
                cell.element.classList.add("mine");
                cell.element.innerText = "💣";
            }
        }
    }
}

```

```

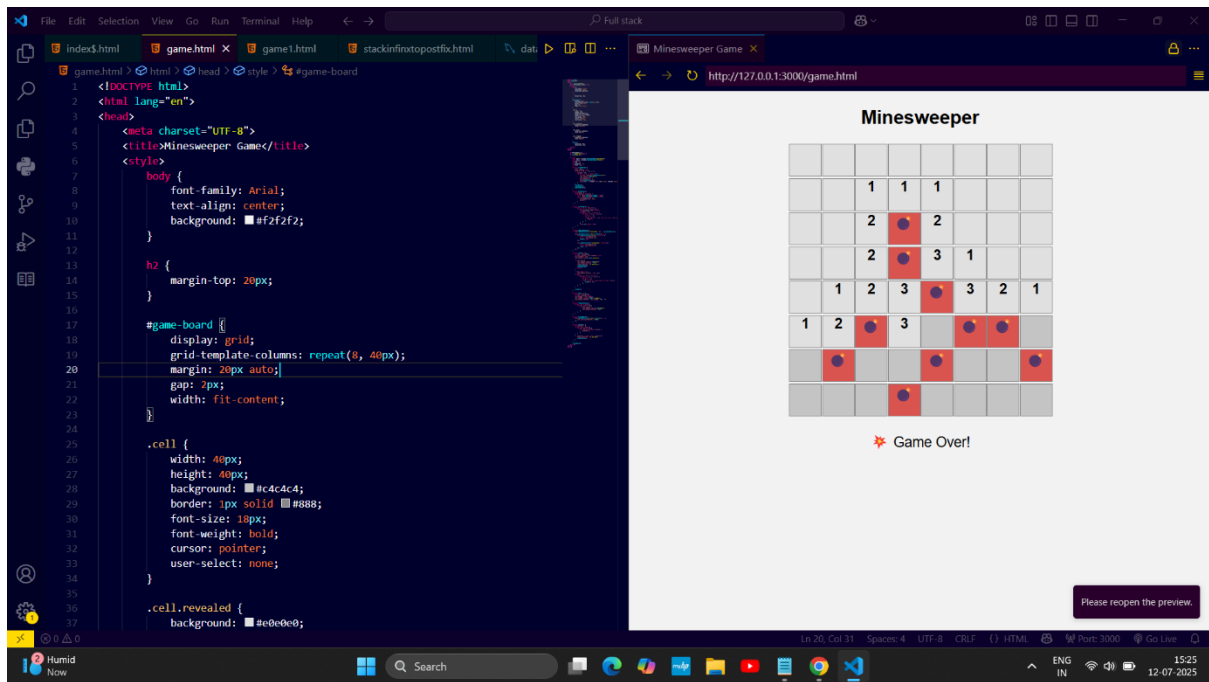
function disableBoard() {

```

```
board.querySelectorAll(".cell").forEach(cell => {  
    cell.style.pointerEvents = "none";  
});  
}
```

```
function checkWin() {  
    let safeCells = 0;  
    for (let row of cells) {  
        for (let cell of row) {  
            if (!cell.mine && cell.revealed) {  
                safeCells++;  
            }  
        }  
    }  
    if (safeCells === rows * cols - mineCount) {  
        status.innerText = "🎉 You Win!";  
        disableBoard();  
    }  
}
```

```
createBoard();  
</script>  
</body>  
</html>
```

7. Full Code (Python) The complete Python code for the console-based Minesweeper game is provided below:

```
import random
```

ROWS, COLS = 8, 8

MINES = 10

```
def create_board():
```

```
board = [{'mine': False, 'revealed': False, 'count': 0, 'flag': False} for _ in range(COLS)] for _ in range(ROWS)]
```

placed = 0

```
while placed < MINES:
```

```
r, c = random.randint(0, ROWS - 1), random.randint(0, COLS - 1)
```

```
if not board[r][c]['mine']:
```

```
board[r][c]['mine'] = True
```

placed += 1

return board

```

def count_adjacent(board):
    for r in range(ROWS):
        for c in range(COLS):
            if board[r][c]['mine']:
                continue
            count = 0
            for i in range(-1, 2):
                for j in range(-1, 2):
                    nr, nc = r + i, c + j
                    if 0 <= nr < ROWS and 0 <= nc < COLS and board[nr][nc]['mine']:
                        count += 1
            board[r][c]['count'] = count

```

```

def reveal(board, r, c):
    if board[r][c]['revealed'] or board[r][c]['flag']:
        return
    board[r][c]['revealed'] = True
    if board[r][c]['count'] == 0 and not board[r][c]['mine']:
        for i in range(-1, 2):
            for j in range(-1, 2):
                nr, nc = r + i, c + j
                if 0 <= nr < ROWS and 0 <= nc < COLS:
                    reveal(board, nr, nc)

```

```

def print_board(board, show_mines=False):
    print(" " + " ".join(str(i) for i in range(COLS)))
    for r in range(ROWS):
        row = f"{r} | "
        for c in range(COLS):
            cell = board[r][c]
            if show_mines and cell['mine']:

```

```

        row += "💣 "
    elif cell['flag']:
        row += "🚩 "
    elif not cell['revealed']:
        row += "# "
    elif cell['mine']:
        row += "💣 "
    elif cell['count'] > 0:
        row += f"{cell['count']} "
    else:
        row += ". "
    print(row)

```

```

def check_win(board):
    for r in range(ROWS):
        for c in range(COLS):
            cell = board[r][c]
            if not cell['mine'] and not cell['revealed']:
                return False
    return True

```

```

def play_game():
    board = create_board()
    count_adjacent(board)
    while True:
        print_board(board)
        action = input("Enter action (r c to reveal, f r c to flag): ").split()
        if len(action) == 2:
            r, c = int(action[0]), int(action[1])
            if board[r][c]['mine']:
                print("\n💣 Game Over!\n")

```

```

        print_board(board, show_mines=True)

        break

    else:

        reveal(board, r, c)

elif len(action) == 3 and action[0].lower() == 'f':

    r, c = int(action[1]), int(action[2])

    board[r][c]['flag'] = not board[r][c]['flag']

if check_win(board):

    print_board(board)

    print("\n🎉 You Win!\n")

    break

```

play_game()

```

1 import random
2
3 ROWS, COLS = 8, 8
4 MINES = 10
5
6 def create_board():
7     board = [{'mine': False, 'revealed': False, 'count': 0, 'flag': False} for _ in
8               range(COLS)] for _ in range(ROWS)]
9     placed = 0
10    while placed < MINES:
11        r, c = random.randint(0, ROWS - 1), random.randint(0, COLS - 1)
12        if not board[r][c]['mine']:
13            board[r][c]['mine'] = True
14            placed += 1
15    return board
16
17 def count_adjacent(board):
18     for r in range(ROWS):
19         for c in range(COLS):
20             if board[r][c]['mine']:
21                 continue
22             count = 0
23             for i in range(-1, 2):
24                 for j in range(-1, 2):
25                     nr, nc = r + i, c + j
26                     if 0 <= nr < ROWS and 0 <= nc < COLS and board[nr][nc]['mine']:
27                         count += 1
28             board[r][c]['count'] = count
29
30 def reveal(board, r, c):
31     if board[r][c]['revealed'] or board[r][c]['flag']:
32         return
33     board[r][c]['revealed'] = True
34     if board[r][c]['count'] == 0 and not board[r][c]['mine']:
35         for i in range(-1, 2):
36             for j in range(-1, 2):
37                 nr, nc = r + i, c + j
38                 if 0 <= nr < ROWS and 0 <= nc < COLS:
39                     reveal(board, nr, nc)
40
41 def print_board(board, show_mines=False):
42     print(" " + " ".join(str(i) for i in range(COLS)))
43     for r in range(ROWS):

```

Enter Input here

If your code takes input, add it in the above box before running.

Output

Status: Runtime error [AI Explain](#)

Time: 0.0300 secs Memory: 11.732 Mb

Your Output

```

0 | # # # # #
1 | # # # # #
2 | # # # # #
3 | # # # # #
4 | # # # # #
5 | # # # # #
6 | # # # # #
7 | # # # # #
Enter action (r c to reveal, f r c to flag):

```

Runtime Error

SIGSEGV

10. Conclusion This project helped me explore the use of JavaScript to simulate event handling and recursive logic in a real-world web application, and Python for a console-based implementation. The game successfully demonstrates movement through logic and user-driven event triggers within a 2D interactive grid, showcasing the versatility of different programming paradigms for the same game concept.

