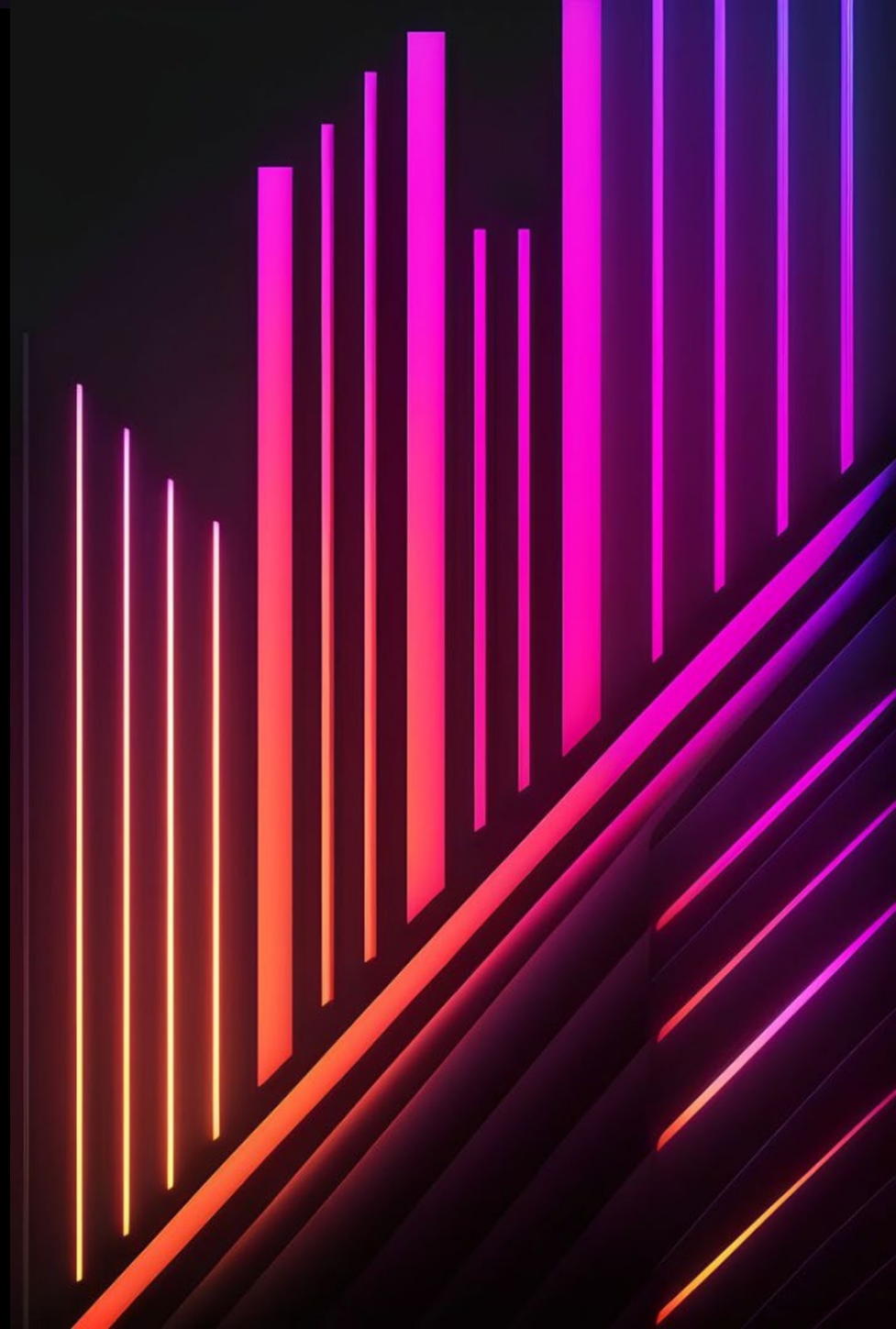
Artificial Intelligence Project

Minesweeper Game



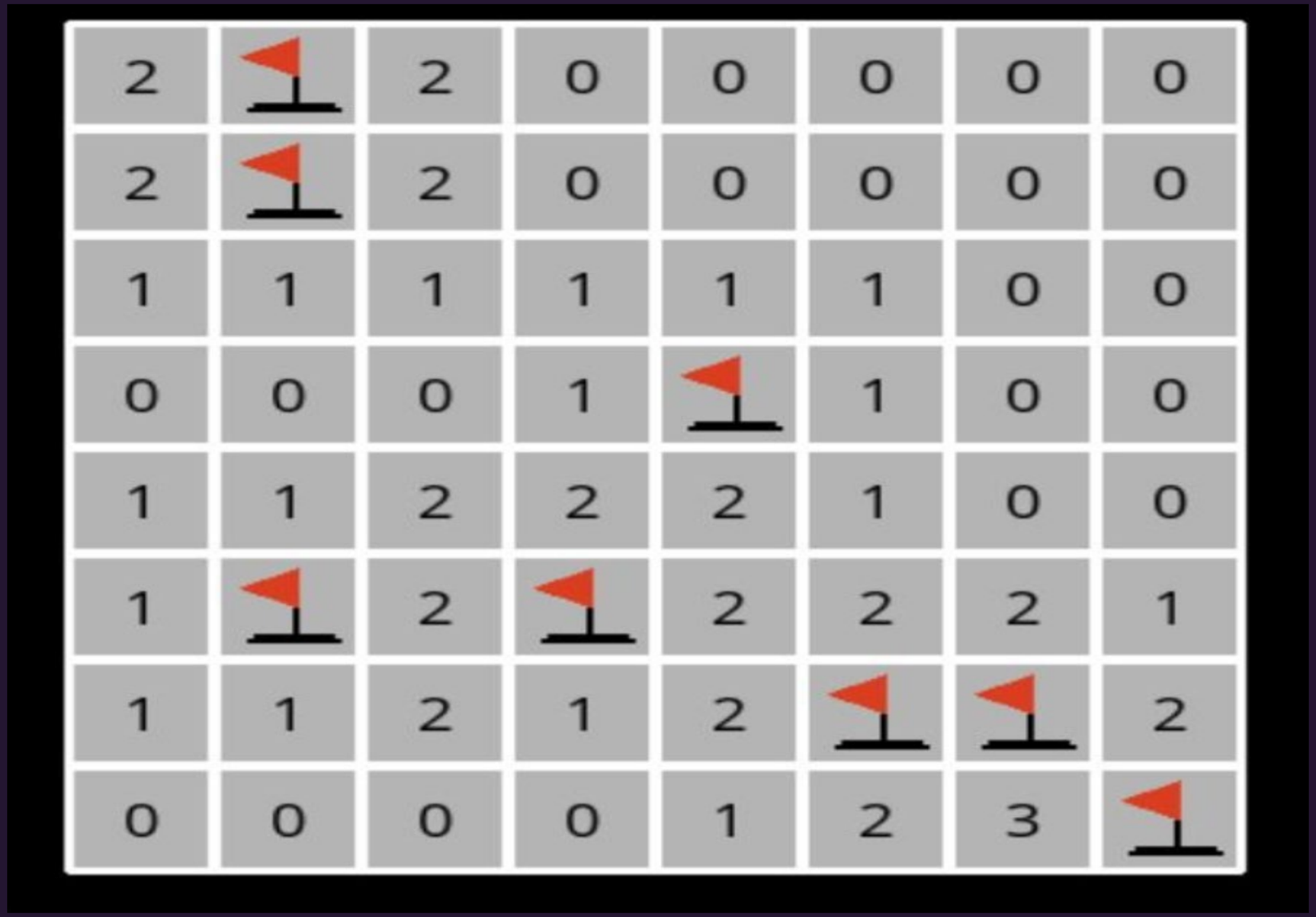
Explanation Of Game

Minesweeper is a puzzle game that consists of a grid of cells, where some of the cells contain hidden “mines.” Clicking on a cell that contains a mine detonates the mine, and causes the user to lose the game. Clicking on a “safe” cell (i.e., a cell that does not contain a mine) reveals a number that indicates how many neighboring cells – where a neighbor is a cell that is one square to the left, right, up, down, or diagonal from the given cell – contain a mine.

The goal of the game is to flag (i.e., identify) each of the mines. In many implementations of the game, including the one in this project, the player can flag a mine by right-clicking on a cell (or two-finger clicking, depending on the computer).

the goal in this project will be to build an AI that can play Minesweeper. Recall that knowledge-based agents make decisions by considering their knowledge base

What information does the AI have access to? Well, the AI would know every time a safe cell is clicked on and would get to see the number for that cell


2 2 2 0 0 0 0 0
2 2 2 0 0 0 0 0
1 1 1 1 1 1 0 0
0 0 0 1 1 0 0
1 1 2 2 2 1 0 0
1 2 2 2 2 2 1
1 1 2 1 2 2
0 0 0 0 1 2 3

Explanation Of Codes

```python
import random

class MinesweeperSolver:
    def __init__(self, rows, cols, mines):
        self.rows = rows
        self.cols = cols
        self.mines = mines
        self.board = [[' ' for _ in range(cols)] for _ in range(rows)]
        self.visited = [[False for _ in range(cols)] for _ in range(rows)]
        self.generate_mines()
```

The first line imports the random module, which is used later to generate random mine positions.

Then we defined a class MinesweeperSolver with an *init* method. The *init* method initializes the solver with the specified number of rows, columns, and mines. It creates a 2D list (self.board) to represent the Minesweeper board, where each cell is initially set to ' '. The self.visited list keeps track of whether a cell has been visited during the solving process. The generate_mines method randomly places mines on the board.

```python
def generate_mines(self):
    mine_positions = random.sample(range(self.rows * self.cols), self.mines)
    for pos in mine_positions:
        row = pos // self.cols
        col = pos % self.cols
        self.board[row][col] = 'M'
```

The method generates random mine positions on the board using the `random.sample` function. It then updates the board to mark the cells containing mines with 'M'.

```python
def is_valid(self, x, y):
    return 0 <= x < self.rows and 0 <= y < self.cols
```

This method checks if a given (x, y) position is valid within the board boundaries.

```python
def count_adjacent_mines(self, x, y):
    count = 0
    for i in range(-1, 2):
        for j in range(-1, 2):
            nx, ny = x + i, y + j
            if self.is_valid(nx, ny) and self.board[nx][ny] == 'M':
                count += 1
    return count
```

This method calculates the number of mines adjacent to a given cell (x, y).

```
def dfs(self, x, y):
    if not self.is_valid(x, y) or self.visited[x][y]:
        return

    self.visited[x][y] = True
    mine_count = self.count_adjacent_mines(x, y)

    if mine_count > 0:
        # Flag cells with mines nearby
        print(f"Flag cell at ({x}, {y}) with {mine_count} mines nearby")
    else:
        # No mines nearby, reveal the cell
        print(f"Reveal cell at ({x}, {y}) with 0 mines nearby")

        for i in range(-1, 2):
            for j in range(-1, 2):
                self.dfs(x + i, y + j)
```

This block of code is part of a depth-first search (DFS) algorithm used to explore and reveal cells on a Minesweeper board. Let's break down the functionality:

The overall effect is that the DFS algorithm recursively explores and reveals cells on the Minesweeper board, flagging cells with mines nearby and revealing cells with no mines nearby. The recursive exploration ensures that the algorithm covers the entire connected region of safe cells.

```
    def solve(self):
        for i in range(self.rows):
            for j in range(self.cols):
                if not self.visited[i][j]:
                    self.dfs(i, j)
```

The solve method iterates through all cells on the board and applies the DFS algorithm to reveal cells that haven't been visited yet.

```
# Example usage:
rows, cols, mines = 10, 5, 1  # Replace with your desired dimensions and number of mines
solver = MinesweeperSolver(rows, cols, mines)
solver.solve()
```

This part creates an instance of the MinesweeperSolver class with the specified dimensions and number of mines and then calls the solve method to reveal cells on the Minesweeper board. The printed output shows information about each revealed cell and the number of adjacent mines.

Thanks