SSN COLLEGE OF ENGINEERING
KALAVAKKAM-603110

Department of Computer Science and Engineering

UCS2504 – Foundations of Artificial Intelligence-CSE-B-23

III Year CSE - (V Semester)

MineSweeper Game

Academic Year 2023-24

Batch: 2021- 2025

Faculty Incharge : Saritha Madhavan

Project Students:

Rohith M	3122 21 5001 085
Rohit Ram	3122 21 5001 086
A.Sadhana	3122 21 5001 089
V.Sanjhay	3122 21 5001 093

INDEX

Problem Statement:

Develop a Minesweeper game and an AI player. The game involves uncovering cells on a grid while avoiding hidden mines. The player must intelligently make moves by deducing safe cells, marking mines, and inferring mine placements based on revealed information to navigate the grid and win the game.

The AI player should showcase strategic decision-making, utilizing logical reasoning to maximize the chances of success in uncovering safe cells while avoiding mines.

A brief overview of the MineSweeper game's rules:

The MineSweeper game typically consists of the following elements:

- Grid: The game is played on a grid of cells, where each cell can be in one of three states: unrevealed, revealed, or flagged.

- Mines: Some of the cells contain hidden mines. The locations of these mines are randomly determined at the start of the game.

- Numbers: Each unrevealed cell adjacent to a mine contains a number indicating the count of mines in its neighboring cells (including diagonals).

- Uncovering Cells: The player can click on unrevealed cells to uncover them. If a revealed cell contains a mine, the game is lost. If it contains a number, that number is shown.

- Flagging Cells: The player can flag cells they suspect contain mines to avoid accidental clicks. Flags do not affect the game's outcome but serve as a visual aid for the player.

- Winning: The player wins by uncovering all safe cells without clicking on any mine.

Software Requirements:

The software requirements are:

1. Programming Language: Python (as used in the provided code).
2. User Interface: For a graphical version, a library like Pygame or Tkinter could be used to create a visual representation of the game board.
3. Documentation: Clear code comments, function docstrings, and an overall README file for users to understand the game and AI logic.
4. Code Editor/IDE: A code editor or integrated development environment (IDE) is useful for writing and running Python code efficiently.

Design Alternatives:

The design alternatives are :

1. AI Strategies: Different AI strategies can be implemented, such as probabilistic reasoning, more advanced logical deduction, or machine learning-based approaches to improve the AI's decision-making process.

2. Board Representation: The game board could be represented using different data structures (e.g., arrays, matrices, dictionaries) for optimized memory usage or improved computational efficiency.

3. Scalability: Design considerations could be made to handle larger board sizes or a variable number of mines, making the game more flexible and adjustable to different difficulty levels.

Suitable Algorithms:

1. Recursive Backtracking: Used to generate the Minesweeper grid by placing mines and updating adjacent cell counts.

2. Logical Deduction: Employed by the AI to infer safe cells and mine locations based on revealed information.

3. Propositional logic:
    a. Atomic Sentences: Cells in the Minesweeper grid are represented as atomic sentences. These sentences denote the presence or absence of mines in specific cells.
    b. Logical Connectives: Logical connectives like AND, OR, and NOT are used to express relationships between cells and their mine status. For example, if cell A has 3 neighboring mines and cell B has 2 neighboring mines, then "A has 3 mines OR B has 2 mines" is represented logically.
    c. Inference Rules: Rules of propositional logic (like modus Ponens, modus Tollens, etc.) are applied to infer new knowledge from existing information. For instance, if a sentence states that out of 8 neighboring cells, 3 contain mines, deductions can be made about the remaining cells.
    d. Implications: If a sentence implies that certain cells must contain mines or must be safe, those deductions are made using logical implications derived from the known information.

4.Decision Making:
    i. Knowledge Base: The AI maintains a knowledge base consisting of logical sentences representing cells and mine counts. It manipulates and updates this knowledge base based on deductions made during the game.
    ii. Deducing Safe Moves: Using logical deductions, the AI infers cells that are definitively safe to uncover and marks them as such.
    iii. Identifying Mines: Through logical reasoning, the AI identifies cells that must contain mines and flags them accordingly, reducing the risk for the player.

Test Cases and Validation:

1. Game Initialization and Basic Functionality:
Test Case 1.1: Default Game Initialization

➔ Description: Create a Minesweeper game with default dimensions and mines.
➔ Expected Outcome: The game should initialize with the default 8x8 grid and 8 mines.

Test Case 1.2: Custom Game Initialization
➔ Description: Create a Minesweeper game with custom dimensions and mine count.
➔ Expected Outcome: The game should initialize with the specified dimensions and mine count.

Test Case 1.3: Print Initial Board
➔ Description: Print the initial state of the Minesweeper board.
➔ Expected Outcome: The board should be displayed with covered cells and mine count indicators.

2. Player Moves:
Test Case 2.1: Uncover Safe Cell
➔ Description: Make a move to uncover a cell without a mine.
➔ Expected Outcome: The cell should be revealed, and the nearby mine count should be displayed.

Test Case 2.2: Uncover Cell with Mine
➔ Description: Make a move to uncover a cell with a mine.
➔ Expected Outcome: The game should end, indicating that the player has hit a mine.

Test Case 2.3: Flag a Mine
➔ Description: Flag a cell as a mine.
➔ Expected Outcome: The flagged cell should be marked, and the player should be able to continue making moves.

Test Case 2.4: Invalid Move
➔ Description: Attempt an invalid move (e.g., flagging an already revealed cell).

➔ Expected Outcome: The game should handle the invalid move gracefully, possibly prompting the player for a valid move.

3. MinesweeperAI Moves:

Test Case 3.1: AI Makes Safe Move

➔ Description: Trigger the MinesweeperAI to make a safe move.

➔ Expected Outcome: The AI should make a move on a cell known to be safe.

Test Case 3.2: AI Makes Random Move

➔ Description: Trigger the MinesweeperAI to make a random move.

➔ Expected Outcome: The AI should make a move on an unrevealed cell, possibly discovering new information.

Test Case 3.3: AI Hits Mine

➔ Description: MinesweeperAI can hit a mine due to lack of information

➔ Expected Outcome: The game should end, indicating that the AI has hit a mine.

4. Game Over Conditions:

Test Case 4.1: Player Wins

➔ Description: Reveal all safe cells and flag all mines correctly.

➔ Expected Outcome: The game should end, indicating that the player has won.

Test Case 4.2: AI Makes Invalid Move

➔ Description: Trigger the MinesweeperAI to make an invalid move.

➔ Expected Outcome: The game should handle the AI's invalid move gracefully, possibly ending the game.

Test Case 4.3: AI Uncovers All Safe Cells

➔ Description: Force the MinesweeperAI to uncover all safe cells.

➔ Expected Outcome: The game should end, indicating that the AI has successfully uncovered all safe cells.

5. Logical Reasoning (MinesweeperAI):

Test Case 5.1: MinesweeperAI Applies Logic based on Knowledge Base

- ➔ Description: Create a scenario where the MinesweeperAI applies advanced logical reasoning to uncover multiple safe cells.
- ➔ Expected Outcome: The AI should make strategic moves based on revealed cells and mine counts.

Test Case 5.2: MinesweeperAI Flags Mines internally

- ➔ Description: Create a scenario where the MinesweeperAI successfully flags all mines.
- ➔ Expected Outcome: The AI should strategically flag cells known to be mines.

These test cases cover a range of scenarios, ensuring that the Minesweeper game and MinesweeperAI behave as expected in various situations. The tests aim to validate the correctness, robustness, and logical reasoning capabilities of the implemented code.

Design

1. Minesweeper Module (minesweeper.py)- Minesweeper class:

This module defines the Minesweeper class, representing the Minesweeper game. It includes methods to initialize the game board, print the board's current state, check if a cell contains a mine, count nearby mines, and determine if the player has won. Additionally, it defines the Sentence class, which represents logical statements about the Minesweeper game. The Sentence class includes methods to determine known mines and safes and update knowledge based on new information.

Attributes:

- height, width: Dimensions of the game board.
- mines: A set containing mine coordinates.
- board: 2D list representing the game state.
- mines_found: Set of cells flagged as mines by the player.

Methods:

1. __init__(self, height=8, width=8, mines=8): Initializes the game with specified dimensions and mine count.
2. print(self): Prints the current game board state.
3. is_mine(self, cell): Checks if a cell contains a mine.
4. nearby_mines(self, cell): Counts adjacent mines for a given cell.
5. won(self): Checks if the player has correctly flagged all mines.

2. MinesweeperAI Module (minesweeper.py)- MinesweeperAI class:
This module defines the MinesweeperAI class, representing the AI player for the Minesweeper game. It includes methods to mark mines and safes, add knowledge based on revealed cells and mine counts, make safe moves, and make random moves. The AI uses logical reasoning to update its knowledge base and make informed decisions about safe moves.

Attributes:

- height, width: Dimensions of the game board.
- moves_made: Set of cells the AI has clicked on.
- mines: Set of cells known to be mines.
- safes: Set of cells known to be safe.
- flags: Set of cells flagged as mines by the AI.
- knowledge: List of Sentence objects representing AI's logical knowledge.

Methods:

1. __init__(self, height=8, width=8): Initializes the AI with specified dimensions.
2. mark_mine(self, cell): Marks a cell as a mine and updates AI's knowledge.
3. mark_safe(self, cell): Marks a cell as safe and updates AI's knowledge.
4. add_knowledge(self, cell, count): Updates knowledge based on a revealed safe cell and its mine count.
5. make_safe_move(self): Returns a safe cell for the AI to choose.
6. make_random_move(self): Returns a random move for the AI to make.

3. Runner Module (runner.py):
This module acts as the interface to run the Minesweeper game. It initializes the game and the AI, provides a function to print the current board state, and runs the game loop. The game loop allows the player to make moves, allows the AI to make moves (either safe or random), and checks for game over conditions. The loop continues until the player either exits the game, the AI makes an invalid move, the player hits a mine, or the player wins.

Functions:

- print_board(): Prints the current game board state.

Game Loop:

1. Initializes the game and AI.
2. Allows the player to make moves (click on a cell or flag a mine).
3. Allows the AI to make moves based on logical reasoning.
4. Checks for game over conditions (hitting a mine, winning) in each iteration.
5. Continues until the player exits, the AI makes an invalid move, or the game reaches a terminal state.

Implementation

MineSweeper, MineSweeperAI, Sentence classes:

```
import itertools
import random


class Minesweeper():
    """Minesweeper game representation"""

    def __init__(self, height=8, width=8, mines=8):

        # Set initial width, height, and number of mines
        self.height = height
        self.width = width
```

```
        self.mines = set()

        # Initialize an empty field with no mines
        self.board = []
        for i in range(self.height):
            row = []
            for j in range(self.width):
                row.append(False)
            self.board.append(row)

        # Add mines randomly
        while len(self.mines) != mines:
            i = random.randrange(height)
            j = random.randrange(width)
            if not self.board[i][j]:
                self.mines.add((i, j))
                self.board[i][j] = True

        # At first, player has found no mines
        self.mines_found = set()

    def print(self):
        #Prints a text-based representation of where mines are
located.
        for i in range(self.height):
            print("--" * self.width + "-")
            for j in range(self.width):
                if (i, j) in ai.moves_made:
                    if (i, j) in ai.flags:
                        print("|F", end="")
                    elif self.board[i][j]:
                        print("|X", end="")
                    else:
                        print(f"|{self.nearby_mines((i, j))}", end="")
                else:
                    print("| ", end="")
            print("|")
        print("--" * self.width + "-")

    def is_mine(self, cell):
```

```
        i, j = cell
        return self.board[i][j]

    def nearby_mines(self, cell):
        #Returns the number of mines that are within one row and
column of a given cell, not including the cell itself.
        # Keep count of nearby mines
        count = 0

        # Loop over all cells within one row and column
        for i in range(cell[0] - 1, cell[0] + 2):
            for j in range(cell[1] - 1, cell[1] + 2):

                # Ignore the cell itself
                if (i, j) == cell:
                    continue

                # Update count if cell in bounds and is mine
                if 0 <= i < self.height and 0 <= j < self.width:
                    if self.board[i][j]:
                        count += 1
        return count

    def won(self):
        #Checks if all mines have been flagged
        return self.mines_found == self.mines


class Sentence():
    #Logical statement about a Minesweeper game, Consists of a set of
board cells, and a count of the number of those cells which are mines.

    def __init__(self, cells, count):
        self.cells = set(cells)
        self.count = count

    def __eq__(self, other):
        return self.cells == other.cells and self.count == other.count

    def __str__(self):
```

```
        return f"{self.cells} = {self.count}"

    def known_mines(self):
        #Returns the set of all cells in self.cells known to be mines.
        if self.count == len(self.cells):
            return self.cells
        return None

    def known_safes(self):
        #Returns the set of all cells in self.cells known to be safe.

        if not self.count:
            return self.cells
        return None

    def mark_mine(self, cell):
        #Updates internal knowledge representation given the fact that
a cell is known to be a mine.
        if cell in self.cells:
            self.cells.remove(cell)
            self.count -= 1

    def mark_safe(self, cell):
        #Updates internal knowledge representation given the fact that
a cell is known to be safe.
        if cell in self.cells:
            self.cells.remove(cell)


class MinesweeperAI():
    """Minesweeper game player"""

    def __init__(self, height=8, width=8):

        # Set initial height and width
        self.height = height
        self.width = width

        # Keep track of which cells have been clicked on
        self.moves_made = set()
```

```
        # Keep track of cells known to be safe or mines
        self.mines = set()
        self.safes = set()
        self.flags = set()

        # List of sentences about the game known to be true
        self.knowledge = []

    def mark_mine(self, cell):
        #Marks a cell as a mine, and updates all knowledge to mark
that cell as a mine as well.
        self.mines.add(cell)
        for sentence in self.knowledge:
            sentence.mark_mine(cell)

    def mark_safe(self, cell):
        #Marks a cell as safe, and updates all knowledge to mark that
cell as safe as well.
        self.safes.add(cell)
        for sentence in self.knowledge:
            sentence.mark_safe(cell)

    def add_knowledge(self, cell, count):
    # Mark the cell as a move that has been made
      self.moves_made.add(cell)

      # Mark the cell as safe
      self.mark_safe(cell)

      # Identify all neighboring cells
      neighbors = set()
      for i in range(max(0, cell[0] - 1), min(self.height, cell[0] +
2)):
          for j in range(max(0, cell[1] - 1), min(self.width, cell[1]
+ 2)):
              if (i, j) != cell and (i, j) not in self.safes:
                  neighbors.add((i, j))

      # Add a new sentence to the AI's knowledge base
```

```
        self.knowledge.append(Sentence(neighbors, count))

        # If count is zero, all neighboring cells are safe
        if count == 0:
            for neighbor in neighbors:
                self.mark_safe(neighbor)

        # Update the AI's knowledge base with new deductions
        self.update_knowledge_base()

    def update_knowledge_base(self):
        # Iterate over all the sentences in the knowledge base
        for sentence in self.knowledge:
            # If a sentence is known to have all its cells as mines
            if sentence.known_mines():
                mines = sentence.known_mines().copy()
                for mine in mines:
                    self.mark_mine(mine)

            # If a sentence is known to have all its cells as safe
            if sentence.known_safes():
                safes = sentence.known_safes().copy()
                for safe in safes:
                    self.mark_safe(safe)

        # Remove empty sentences from the knowledge base
        self.knowledge = [s for s in self.knowledge if len(s.cells) >
0]

    def make_safe_move(self):
        #Returns a safe cell to choose on the Minesweeper board.
        available_steps = self.safes - self.moves_made
        if available_steps:
            return random.choice(tuple(available_steps))
        return None

    def make_random_move(self):
        #Returns a move to make on the Minesweeper board.

        # if no move can be made
```

```
        if len(self.mines) + len(self.moves_made) == self.height *
self.width:
            return None

        # loop until an appropriate move is found
        while True:
            i = random.randrange(self.height)
            j = random.randrange(self.width)
            if (i, j) not in self.moves_made and (i, j) not in
self.mines:
                return (i, j)
```

Runner.py:

```
# from minesweeper import Minesweeper, MinesweeperAI
import time

# Game configuration
HEIGHT, WIDTH, MINES = 10, 10, 20

# Initialize game and AI
game = Minesweeper(height=HEIGHT, width=WIDTH, mines=MINES)
ai = MinesweeperAI(height=HEIGHT, width=WIDTH)
flagged_cells = set()

def print_board():
    """Function to print the current board state to the console."""
    # Print column numbers
    print("   ", end="")
    for j in range(WIDTH):
        print(f"{j:^3}", end="")
    print("\n  " + "----" * WIDTH)

    for i in range(HEIGHT):
        # Print row number
        print(f"{i:^2}|", end="")
        for j in range(WIDTH):
            if (i, j) in flagged_cells:
                print(f"{'F':^3}", end="")
```

```
            elif (i, j) in ai.moves_made:
                if game.is_mine((i, j)):
                    print(f"{'X':^3}", end="")
                else:
                    print(f"{game.nearby_mines((i, j)):^3}", end="")
            else:
                print(f"{'*':^3}", end="")
        print()

    print()


game_over = False

while not game_over:
    print("Minesweeper")
    print_board()

    user_input = input("Enter your move (row col), 'ai' for AI move,
'flag row col' to mark a location, or 'exit' to quit: ")

    if user_input.lower() == 'exit':
        break
    elif user_input.lower() == 'ai':
        move = ai.make_safe_move() or ai.make_random_move()
        if move is None:
            print("No moves left.")
            break
        else:
            print(f"AI move: {move}")
            if game.is_mine(move):
                print("AI hit a mine! Game over.")
                game_over = True
            else:
                ai.add_knowledge(move, game.nearby_mines(move))
    elif user_input.lower().startswith('flag'):
      try:
          _, row, col = user_input.split()
          row, col = int(row), int(col)
          # Add the flagged cell to the set
```

```
            flagged_cells.add((row, col))
        except ValueError:
            print("Invalid input. Please enter a valid move or flag.")

    else:
        try:
            row, col = map(int, user_input.split())
            if game.is_mine((row, col)):
                print("Mine hit! Game over.")
                game_over = True
            else:
                ai.add_knowledge((row, col), game.nearby_mines((row,
col)))
                print(ai.knowledge)
        except ValueError:
            print("Invalid input. Please enter a valid move.")

    if game.won():
        print("Congratulations! You have won!")
        game_over = True

    time.sleep(1)
```

Validation

1. Basic Gameplay:

Manually play the game, ensuring accurate player move responses and correct game termination.

2. Visual Inspection of Game Board:

Print and visually inspect the game board at different stages for expected behavior.

3. Randomized Gameplay:

Verify consistent behavior across varied game setups.

4. AI Decision Validation:

Observe MinesweeperAI moves, confirming logical and reasonable decisions aligned with Minesweeper rules.

5. Boundary Testing:

Test extreme values for board size and mines, checking for performance and ensuring a playable game.

6. User Input Handling:

Input invalid commands or moves, confirming graceful handling and informative error messages.

7. Edge Case Testing:

Test scenarios requiring advanced MinesweeperAI reasoning, confirming adaptation to diverse game states.

Output:

```
Minesweeper
    0  1  2  3  4  5  6  7  8  9
  ---------------------------------------
0 |  *  *  *  *  *  *  *  *  *  *
1 |  *  *  *  *  *  *  *  *  *  *
2 |  *  *  *  *  *  *  *  *  *  *
3 |  *  *  *  *  *  *  *  *  *  *
4 |  *  *  *  *  *  *  *  *  *  *
5 |  *  *  *  *  *  *  *  *  *  *
6 |  *  *  *  *  *  *  *  *  *  *
7 |  *  *  *  *  *  *  *  *  *  *
8 |  *  *  *  *  *  *  *  *  *  *
9 |  *  *  *  *  *  *  *  *  *  *

Enter your move (row col), 'ai' for AI move, 'flag row col' to mark a location, or 'exit' to quit: ai
AI move: (5, 2)
Minesweeper
    0  1  2  3  4  5  6  7  8  9
  ---------------------------------------
0 |  *  *  *  *  *  *  *  *  *  *
1 |  *  *  *  *  *  *  *  *  *  *
2 |  *  *  *  *  *  *  *  *  *  *
3 |  *  *  *  *  *  *  *  *  *  *
4 |  *  *  *  *  *  *  *  *  *  *
5 |  *  *  1  *  *  *  *  *  *  *
6 |  *  *  *  *  *  *  *  *  *  *
7 |  *  *  *  *  *  *  *  *  *  *
8 |  *  *  *  *  *  *  *  *  *  *
9 |  *  *  *  *  *  *  *  *  *  *
```

```
Enter your move (row col), 'ai' for AI move, 'flag row col' to mark a location, or 'exit' to quit: ai
AI move: (8, 0)
Minesweeper
    0  1  2  3  4  5  6  7  8  9
  ----------------------------------------
0 | *  *  *  *  *  *  *  *  *  *
1 | *  *  *  *  *  *  *  *  *  *
2 | *  *  *  *  *  *  *  *  *  *
3 | *  *  *  *  *  *  *  *  *  *
4 | *  *  *  *  *  *  *  *  *  *
5 | *  *  1  *  *  *  *  *  *  *
6 | *  *  *  *  *  *  *  *  *  *
7 | *  *  *  *  *  *  *  *  *  *
8 | 1  *  *  *  *  *  *  *  *  *
9 | *  *  *  *  *  *  *  *  *  *

Enter your move (row col), 'ai' for AI move, 'flag row col' to mark a location, or 'exit' to quit: ai
AI move: (4, 0)
Minesweeper
    0  1  2  3  4  5  6  7  8  9
  ----------------------------------------
0 | *  *  *  *  *  *  *  *  *  *
1 | *  *  *  *  *  *  *  *  *  *
2 | *  *  *  *  *  *  *  *  *  *
3 | *  *  *  *  *  *  *  *  *  *
4 | 1  *  *  *  *  *  *  *  *  *
5 | *  *  1  *  *  *  *  *  *  *
6 | *  *  *  *  *  *  *  *  *  *
7 | *  *  *  *  *  *  *  *  *  *
8 | 1  *  *  *  *  *  *  *  *  *
9 | *  *  *  *  *  *  *  *  *  *
```

```
Enter your move (row col), 'ai' for AI move, 'flag row col' to mark a location, or 'exit' to quit: ai
AI move: (6, 0)
Minesweeper
    0  1  2  3  4  5  6  7  8  9
  ----------------------------------------
0 | *  *  *  *  *  *  *  *  *  *
1 | *  *  *  *  *  *  *  *  *  *
2 | *  *  *  *  *  *  *  *  *  *
3 | *  *  *  *  *  *  *  *  *  *
4 | 1  *  *  *  *  *  *  *  *  *
5 | *  *  1  *  *  *  *  *  *  *
6 | 0  *  *  *  *  *  *  *  *  *
7 | *  *  *  *  *  *  *  *  *  *
8 | 1  *  *  *  *  *  *  *  *  *
9 | *  *  *  *  *  *  *  *  *  *

Enter your move (row col), 'ai' for AI move, 'flag row col' to mark a location, or 'exit' to quit: ai
AI move: (5, 0)
Minesweeper
    0  1  2  3  4  5  6  7  8  9
  ----------------------------------------
0 | *  *  *  *  *  *  *  *  *  *
1 | *  *  *  *  *  *  *  *  *  *
2 | *  *  *  *  *  *  *  *  *  *
3 | *  *  *  *  *  *  *  *  *  *
4 | 1  *  *  *  *  *  *  *  *  *
5 | 0  *  1  *  *  *  *  *  *  *
6 | 0  *  *  *  *  *  *  *  *  *
7 | *  *  *  *  *  *  *  *  *  *
8 | 1  *  *  *  *  *  *  *  *  *
9 | *  *  *  *  *  *  *  *  *  *
```

```
Enter your move (row col), 'ai' for AI move, 'flag row col' to mark a location, or 'exit' to quit: ai
AI move: (6, 1)
Minesweeper
    0  1  2  3  4  5  6  7  8  9
  ----------------------------------------
0 | *  *  *  *  *  *  *  *  *  *
1 | *  *  *  *  *  *  *  *  *  *
2 | *  *  *  *  *  *  *  *  *  *
3 | *  *  *  *  *  *  *  *  *  *
4 | 1  *  *  *  *  *  *  *  *  *
5 | 0  *  1  *  *  *  *  *  *  *
6 | 0  0  *  *  *  *  *  *  *  *
7 | *  *  *  *  *  *  *  *  *  *
8 | 1  *  *  *  *  *  *  *  *  *
9 | *  *  *  *  *  *  *  *  *  *

Enter your move (row col), 'ai' for AI move, 'flag row col' to mark a location, or 'exit' to quit: ai
AI move: (6, 2)
Minesweeper
    0  1  2  3  4  5  6  7  8  9
  ----------------------------------------
0 | *  *  *  *  *  *  *  *  *  *
1 | *  *  *  *  *  *  *  *  *  *
2 | *  *  *  *  *  *  *  *  *  *
3 | *  *  *  *  *  *  *  *  *  *
4 | 1  *  *  *  *  *  *  *  *  *
5 | 0  *  1  *  *  *  *  *  *  *
6 | 0  0  0  *  *  *  *  *  *  *
7 | *  *  *  *  *  *  *  *  *  *
8 | 1  *  *  *  *  *  *  *  *  *
9 | *  *  *  *  *  *  *  *  *  *
```

```
Enter your move (row col), 'ai' for AI move, 'flag row col' to mark a location, or 'exit' to quit: ai
AI move: (7, 0)
Minesweeper
    0  1  2  3  4  5  6  7  8  9
  ----------------------------------------
0 | *  *  *  *  *  *  *  *  *  *
1 | *  *  *  *  *  *  *  *  *  *
2 | *  *  *  *  *  *  *  *  *  *
3 | *  *  *  *  *  *  *  *  *  *
4 | 1  *  *  *  *  *  *  *  *  *
5 | 0  *  1  *  *  *  *  *  *  *
6 | 0  0  0  *  *  *  *  *  *  *
7 | 0  *  *  *  *  *  *  *  *  *
8 | 1  *  *  *  *  *  *  *  *  *
9 | *  *  *  *  *  *  *  *  *  *

Enter your move (row col), 'ai' for AI move, 'flag row col' to mark a location, or 'exit' to quit: ai
AI move: (7, 2)
Minesweeper
    0  1  2  3  4  5  6  7  8  9
  ----------------------------------------
0 | *  *  *  *  *  *  *  *  *  *
1 | *  *  *  *  *  *  *  *  *  *
2 | *  *  *  *  *  *  *  *  *  *
3 | *  *  *  *  *  *  *  *  *  *
4 | 1  *  *  *  *  *  *  *  *  *
5 | 0  *  1  *  *  *  *  *  *  *
6 | 0  0  0  *  *  *  *  *  *  *
7 | 0  *  0  *  *  *  *  *  *  *
8 | 1  *  *  *  *  *  *  *  *  *
9 | *  *  *  *  *  *  *  *  *  *
```

```
Enter your move (row col), 'ai' for AI move, 'flag row col' to mark a location, or 'exit' to quit: ai
AI move: (8, 2)
Minesweeper
    0  1  2  3  4  5  6  7  8  9
  ---------------------------------------
0 | *  *  *  *  *  *  *  *  *  *
1 | *  *  *  *  *  *  *  *  *  *
2 | *  *  *  *  *  *  *  *  *  *
3 | *  *  *  *  *  *  *  *  *  *
4 | 1  *  *  *  *  *  *  *  *  *
5 | 0  *  1  *  *  *  *  *  *  *
6 | 0  0  0  *  *  *  *  *  *  *
7 | 0  *  0  *  *  *  *  *  *  *
8 | 1  *  1  *  *  *  *  *  *  *
9 | *  *  *  *  *  *  *  *  *  *

Enter your move (row col), 'ai' for AI move, 'flag row col' to mark a location, or 'exit' to quit: ai
AI move: (8, 3)
Minesweeper
    0  1  2  3  4  5  6  7  8  9
  ---------------------------------------
0 | *  *  *  *  *  *  *  *  *  *
1 | *  *  *  *  *  *  *  *  *  *
2 | *  *  *  *  *  *  *  *  *  *
3 | *  *  *  *  *  *  *  *  *  *
4 | 1  *  *  *  *  *  *  *  *  *
5 | 0  *  1  *  *  *  *  *  *  *
6 | 0  0  0  *  *  *  *  *  *  *
7 | 0  *  0  *  *  *  *  *  *  *
8 | 1  *  1  1  *  *  *  *  *  *
9 | *  *  *  *  *  *  *  *  *  *
```

```
Enter your move (row col), 'ai' for AI move, 'flag row col' to mark a location, or 'exit' to quit: ai
AI move: (7, 1)
Minesweeper
    0  1  2  3  4  5  6  7  8  9
  ---------------------------------------
0 | *  *  *  *  *  *  *  *  *  *
1 | *  *  *  *  *  *  *  *  *  *
2 | *  *  *  *  *  *  *  *  *  *
3 | *  *  *  *  *  *  *  *  *  *
4 | 1  *  *  *  *  *  *  *  *  *
5 | 0  *  1  *  *  *  *  *  *  *
6 | 0  0  0  *  *  *  *  *  *  *
7 | 0  0  0  *  *  *  *  *  *  *
8 | 1  *  1  1  *  *  *  *  *  *
9 | *  *  *  *  *  *  *  *  *  *

Enter your move (row col), 'ai' for AI move, 'flag row col' to mark a location, or 'exit' to quit: ai
AI move: (8, 1)
Minesweeper
    0  1  2  3  4  5  6  7  8  9
  ---------------------------------------
0 | *  *  *  *  *  *  *  *  *  *
1 | *  *  *  *  *  *  *  *  *  *
2 | *  *  *  *  *  *  *  *  *  *
3 | *  *  *  *  *  *  *  *  *  *
4 | 1  *  *  *  *  *  *  *  *  *
5 | 0  *  1  *  *  *  *  *  *  *
6 | 0  0  0  *  *  *  *  *  *  *
7 | 0  0  0  *  *  *  *  *  *  *
8 | 1  1  1  1  *  *  *  *  *  *
9 | *  *  *  *  *  *  *  *  *  *
```

```
Enter your move (row col), 'ai' for AI move, 'flag row col' to mark a location, or 'exit' to quit: ai
AI move: (4, 1)
Minesweeper
    0  1  2  3  4  5  6  7  8  9
  ----------------------------------------
0 | *  *  *  *  *  *  *  *  *  *
1 | *  *  *  *  *  *  *  *  *  *
2 | *  *  *  *  *  *  *  *  *  *
3 | *  *  *  *  *  *  *  *  *  *
4 | 1  2  *  *  *  *  *  *  *  *
5 | 0  *  1  *  *  *  *  *  *  *
6 | 0  0  0  *  *  *  *  *  *  *
7 | 0  0  0  *  *  *  *  *  *  *
8 | 1  1  1  1  *  *  *  *  *  *
9 | *  *  *  *  *  *  *  *  *  *

Enter your move (row col), 'ai' for AI move, 'flag row col' to mark a location, or 'exit' to quit: ai
AI move: (7, 3)
Minesweeper
    0  1  2  3  4  5  6  7  8  9
  ----------------------------------------
0 | *  *  *  *  *  *  *  *  *  *
1 | *  *  *  *  *  *  *  *  *  *
2 | *  *  *  *  *  *  *  *  *  *
3 | *  *  *  *  *  *  *  *  *  *
4 | 1  2  *  *  *  *  *  *  *  *
5 | 0  *  1  *  *  *  *  *  *  *
6 | 0  0  0  *  *  *  *  *  *  *
7 | 0  0  0  1  *  *  *  *  *  *
8 | 1  1  1  1  *  *  *  *  *  *
9 | *  *  *  *  *  *  *  *  *  *
```

```
Enter your move (row col), 'ai' for AI move, 'flag row col' to mark a location, or 'exit' to quit: ai
AI move: (6, 3)
Minesweeper
    0  1  2  3  4  5  6  7  8  9
  ----------------------------------------
0 | *  *  *  *  *  *  *  *  *  *
1 | *  *  *  *  *  *  *  *  *  *
2 | *  *  *  *  *  *  *  *  *  *
3 | *  *  *  *  *  *  *  *  *  *
4 | 1  2  *  *  *  *  *  *  *  *
5 | 0  *  1  *  *  *  *  *  *  *
6 | 0  0  0  1  *  *  *  *  *  *
7 | 0  0  0  1  *  *  *  *  *  *
8 | 1  1  1  1  *  *  *  *  *  *
9 | *  *  *  *  *  *  *  *  *  *

Enter your move (row col), 'ai' for AI move, 'flag row col' to mark a location, or 'exit' to quit: ai
AI move: (5, 3)
Minesweeper
    0  1  2  3  4  5  6  7  8  9
  ----------------------------------------
0 | *  *  *  *  *  *  *  *  *  *
1 | *  *  *  *  *  *  *  *  *  *
2 | *  *  *  *  *  *  *  *  *  *
3 | *  *  *  *  *  *  *  *  *  *
4 | 1  2  *  *  *  *  *  *  *  *
5 | 0  *  1  2  *  *  *  *  *  *
6 | 0  0  0  1  *  *  *  *  *  *
7 | 0  0  0  1  *  *  *  *  *  *
8 | 1  1  1  1  *  *  *  *  *  *
9 | *  *  *  *  *  *  *  *  *  *
```

```
Enter your move (row col), 'ai' for AI move, 'flag row col' to mark a location, or 'exit' to quit: ai
AI move: (5, 1)
Minesweeper
    0  1  2  3  4  5  6  7  8  9
  ---------------------------------------
0 | *  *  *  *  *  *  *  *  *  *
1 | *  *  *  *  *  *  *  *  *  *
2 | *  *  *  *  *  *  *  *  *  *
3 | *  *  *  *  *  *  *  *  *  *
4 | 1  2  *  *  *  *  *  *  *  *
5 | 0  1  1  2  *  *  *  *  *  *
6 | 0  0  0  1  *  *  *  *  *  *
7 | 0  0  0  1  *  *  *  *  *  *
8 | 1  1  1  1  *  *  *  *  *  *
9 | *  *  *  *  *  *  *  *  *  *

Enter your move (row col), 'ai' for AI move, 'flag row col' to mark a location, or 'exit' to quit: ai
AI move: (8, 4)
Minesweeper
    0  1  2  3  4  5  6  7  8  9
  ---------------------------------------
0 | *  *  *  *  *  *  *  *  *  *
1 | *  *  *  *  *  *  *  *  *  *
2 | *  *  *  *  *  *  *  *  *  *
3 | *  *  *  *  *  *  *  *  *  *
4 | 1  2  *  *  *  *  *  *  *  *
5 | 0  1  1  2  *  *  *  *  *  *
6 | 0  0  0  1  *  *  *  *  *  *
7 | 0  0  0  1  *  *  *  *  *  *
8 | 1  1  1  1  2  *  *  *  *  *
9 | *  *  *  *  *  *  *  *  *  *
```

```
Enter your move (row col), 'ai' for AI move, 'flag row col' to mark a location, or 'exit' to quit: ai
AI move: (4, 3)
Minesweeper
    0  1  2  3  4  5  6  7  8  9
  ---------------------------------------
0 | *  *  *  *  *  *  *  *  *  *
1 | *  *  *  *  *  *  *  *  *  *
2 | *  *  *  *  *  *  *  *  *  *
3 | *  *  *  *  *  *  *  *  *  *
4 | 1  2  *  2  *  *  *  *  *  *
5 | 0  1  1  2  *  *  *  *  *  *
6 | 0  0  0  1  *  *  *  *  *  *
7 | 0  0  0  1  *  *  *  *  *  *
8 | 1  1  1  1  2  *  *  *  *  *
9 | *  *  *  *  *  *  *  *  *  *

Enter your move (row col), 'ai' for AI move, 'flag row col' to mark a location, or 'exit' to quit: ai
AI move: (0, 4)
Minesweeper
    0  1  2  3  4  5  6  7  8  9
  ---------------------------------------
0 | *  *  *  *  1  *  *  *  *  *
1 | *  *  *  *  *  *  *  *  *  *
2 | *  *  *  *  *  *  *  *  *  *
3 | *  *  *  *  *  *  *  *  *  *
4 | 1  2  *  2  *  *  *  *  *  *
5 | 0  1  1  2  *  *  *  *  *  *
6 | 0  0  0  1  *  *  *  *  *  *
7 | 0  0  0  1  *  *  *  *  *  *
8 | 1  1  1  1  2  *  *  *  *  *
9 | *  *  *  *  *  *  *  *  *  *
```

```
Enter your move (row col), 'ai' for AI move, 'flag row col' to mark a location, or 'exit' to quit: flag 4 2
Minesweeper
    0  1  2  3  4  5  6  7  8  9
  ---------------------------------------
0 | *  *  *  *  1  *  *  *  *  *
1 | *  *  *  *  *  *  *  *  *  *
2 | *  *  *  *  *  *  *  *  *  *
3 | *  *  *  *  *  *  *  *  *  *
4 | 1  2  F  2  *  *  *  *  *  *
5 | 0  1  1  2  *  *  *  *  *  *
6 | 0  0  0  1  *  *  *  *  *  *
7 | 0  0  0  1  *  *  *  *  *  *
8 | 1  1  1  1  2  *  *  *  *  *
9 | *  *  *  *  *  *  *  *  *  *

Enter your move (row col), 'ai' for AI move, 'flag row col' to mark a location, or 'exit' to quit: ai
AI move: (1, 2)
Minesweeper
    0  1  2  3  4  5  6  7  8  9
  ---------------------------------------
0 | *  *  *  *  1  *  *  *  *  *
1 | *  *  1  *  *  *  *  *  *  *
2 | *  *  *  *  *  *  *  *  *  *
3 | *  *  *  *  *  *  *  *  *  *
4 | 1  2  F  2  *  *  *  *  *  *
5 | 0  1  1  2  *  *  *  *  *  *
6 | 0  0  0  1  *  *  *  *  *  *
7 | 0  0  0  1  *  *  *  *  *  *
8 | 1  1  1  1  2  *  *  *  *  *
9 | *  *  *  *  *  *  *  *  *  *
```

```
Enter your move (row col), 'ai' for AI move, 'flag row col' to mark a location, or 'exit' to quit: ai
AI move: (5, 5)
Minesweeper
    0  1  2  3  4  5  6  7  8  9
  ---------------------------------------
0 | *  *  *  *  1  *  *  *  *  *
1 | *  *  1  *  *  *  *  *  *  *
2 | *  *  *  *  *  *  *  *  *  *
3 | *  *  *  *  *  *  *  *  *  *
4 | 1  2  F  2  *  *  *  *  *  *
5 | 0  1  1  2  *  4  *  *  *  *
6 | 0  0  0  1  *  *  *  *  *  *
7 | 0  0  0  1  *  *  *  *  *  *
8 | 1  1  1  1  2  *  *  *  *  *
9 | *  *  *  *  *  *  *  *  *  *

Enter your move (row col), 'ai' for AI move, 'flag row col' to mark a location, or 'exit' to quit: ai
AI move: (1, 6)
AI hit a mine! Game over.
```