INDIAN INSTITUTE OF TECHNOLOGY TIRUPATI
DEPARTMENT OF MATHEMATICS AND STATISTICS

Project - 1 **MA517M-Basic Programming Laboratory** Last Date: 09 November 2025

Name **Roll No.: MA25M003**

Matrix Battleship

Objective

The objective of this project is to design and implement a console-based version of the classic **Battleship** game, but represented mathematically using a **matrix grid**. Players must use logical reasoning and coordinate-based strategies to locate hidden "ships" on the matrix by guessing their positions.

This project enables students to:

- Apply the concept of **matrices** and coordinate systems computationally.
- Strengthen problem-solving and pattern-recognition skills.
- Gain hands-on practice with `C++` data structures such as `arrays`, `structures`, and `classes`.

Mathematical Background

A two-dimensional matrix $M \in \{0, 1, 2\}^{n \times n}$ is used to represent the game board:

$$M_{ij} = \begin{cases} 0, & \text{if the cell is empty,} \\ 1, & \text{if the cell contains part of a ship,} \\ 2, & \text{if the cell has been hit or guessed.} \end{cases}$$

Each coordinate (i, j) corresponds to a specific location in the grid. The player aims to find all cells with value 1 (the hidden ships) through successive guesses.

The game can be viewed as a discrete mathematical model of a search problem on a finite grid.

Game Description

- The computer initializes an $n \times n$ matrix (e.g., $n = 5$).
- Randomly, a few cells (say, 3–5) are assigned value 1 to represent ship positions.
- The rest of the matrix cells are initialized to 0.
- The player does not see the full matrix — only the result of their guesses.
- On each turn, the player enters coordinates (i, j).
- The program responds:
  - `HIT!` if $M_{ij} = 1$,

  - `MISS!` if $M_{ij} = 0$,
  - `ALREADY GUESSED!` if $M_{ij} = 2$.
- The cell is then marked as 2 (guessed).
- The game continues until all ships are found or until the maximum number of guesses is reached.

Algorithm

1. Initialize:
   - Matrix size n,
   - Randomly assign a few ship locations.
   - Set all other cells to zero.
2. While ships remain undiscovered and moves $<$ max limit:
   (a) Display partial grid with marks for guessed cells.
   (b) Read player input (i, j).
   (c) If $M_{ij} = 1$, mark it as 2 and increment hits.
   (d) If $M_{ij} = 0$, mark it as 2 and show "MISS!".
   (e) If $M_{ij} = 2$, show "ALREADY GUESSED!".
   (f) Display the updated matrix state.
3. If all ships are found, display "Congratulations, you sank all ships!".
4. Otherwise, display "Game Over".

Program Structure (C++ Skeleton)

```
struct Matrix {
    int grid[5][5];
    int n;
};

class MatrixBattleship {
private:
    Matrix board;
    int numShips;
    int hits;
    int moves;
public:
    void initialize();
    void displayMasked();
    bool guess(int i, int j);
    bool allShipsSunk();
```

```
    void play();
};
```

Sample Interaction

Output:

```
Welcome to Matrix Battleship!

Grid Size: 5 x 5
Total Ships: 3
You have 10 attempts to sink them all.

Enter coordinates (row column): 2 3
MISS!

Enter coordinates (row column): 1 4
HIT!

Current Grid:
[ . . . X . ]
[ . . O . . ]
[ . . . . . ]
[ . . . . . ]
[ . . . . . ]

Ships remaining: 2
Attempts left: 8
```

Legend:

- `.` → unguessed cell
- `O` → missed guess
- `X` → hit

Possible Extensions

- Implement computer-vs-player mode (AI guessing).
- Add multiple ship sizes (occupying adjacent cells).
- Introduce scoring or difficulty levels.
- Use dynamic matrix allocation for variable grid sizes.

Project - 2: 2048 Game Using C++ Classes

Problem Statement

Design and implement the popular **2048 Game** using **C++ classes**. The program should simulate the 4×4 sliding tile game where the player combines numbers by sliding them in four directions to reach the tile with value 2048. The project should utilize object-oriented programming concepts such as classes, objects, encapsulation, and methods for handling game logic and user interaction.

Project Requirements

1. Create a `Board` class to represent the 4×4 game grid.
   (a) Include a method to initialize the board with two random tiles of 2 or 4.
   (b) Include a method to generate a new random tile in an empty position after each move.
   (c) Include a method to display the current board in a nicely formatted way.
   (d) Include a method to check for the game over condition.
2. Create a `Game` class to manage gameplay.
   (a) Display options for the user: `Play` or `Solution`.
   (b) If the user chooses `Play`, show the navigation commands:
      - `W/w` for Up
      - `S/s` for Down
      - `A/a` for Left
      - `D/d` for Right
      - `Q/q` for Quit
   (c) When the user presses `Q/q`, confirm before quitting.
   (d) For each move, merge tiles according to 2048 rules and update the board.
   (e) Display the board after each move.
3. If the user chooses `Solution`, demonstrate a sequence of moves that leads to a high-value tile (e.g., 2048).
4. Ensure proper encapsulation of game logic and board operations within the respective classes.

Suggested Class Structure

1. `Board` Class:
   - Data member: 4×4 integer array representing tiles
   - Methods: initializeBoard(), addRandomTile(), displayBoard(), isGameOver(), mergeTiles(char direction)
2. `Game` Class:
   - Data member: `Board` object, user choice, score
   - Methods: playGame(), showSolution(), processMove(char move), confirmQuit()