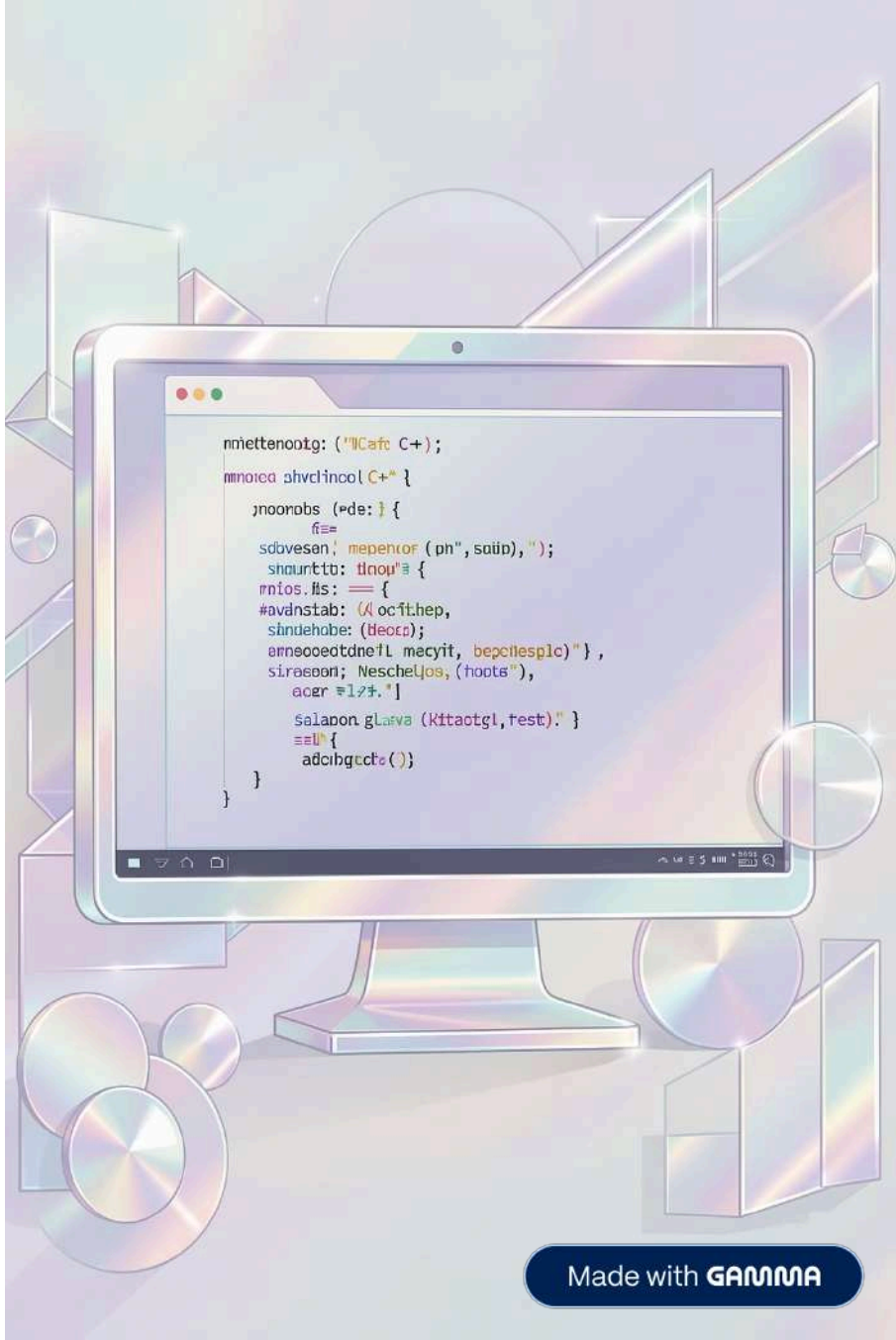
Tic-Tac-Toe Game in C++

Console-Based Game Implementation Using 2D Arrays

C++ PROGRAMMING | FIRST YEAR PROJECT

Presented By:
Aman Koli
Aditya Yadav
Tanmay Sherkar
Shubham Mane

College:
ITM Skills University
B.Tech Computer Science & Engineering
First Year – Batch 2025–2029



Introduction to Tic-Tac-Toe

01

Two-Player Classic

A strategic game where two players compete head-to-head, taking turns to claim spaces on the board.

02

3×3 Grid Arena

The game is played on a simple 3×3 grid, offering 9 possible positions for strategic placement.

03

Alternating Marks

Players alternate marking their symbol—X or O—attempting to outmaneuver their opponent.

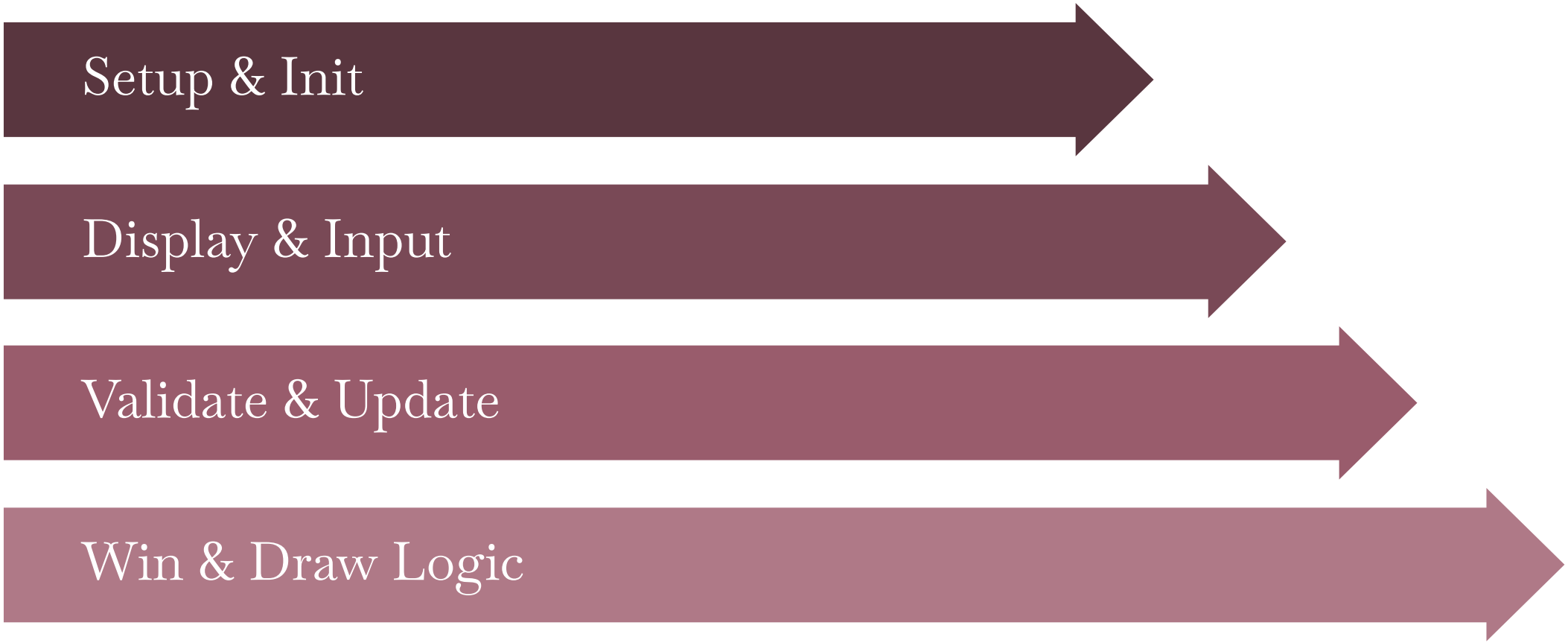
04

Win or Draw

Victory comes from aligning three marks in a row, column, or diagonal. If all positions fill without a winner, the game ends in a draw.

Program Architecture

Our implementation follows a modular approach, dividing the game into four distinct functional components that work together seamlessly.



This structured design ensures clean code organization and makes debugging simpler while maintaining logical separation of concerns.

Part 1: Game Initialization

Include Libraries

iostream for input/output operations

Create Game Board

3×3 character array to store positions

Initialize Positions

Board numbered 1-9 for easy reference

Set Starting Player

Player 'X' begins the game

Track Moves

Counter variable monitors game progress

```
char board[3][3] = {
   {'1','2','3'},
   {'4','5','6'},
   {'7','8','9'}
};

char player = 'X';
int choice, moves = 0;
```

This initialization sets up all essential variables needed to run the game, establishing the foundation for gameplay logic.

Part 2: Board Display & Input

```
while (true) {
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            cout << " " << board[i][j] << " ";
            if (j < 2) cout << "|";
        }
        cout << endl;
        if (i < 2)
            cout << "---|---|---" << endl;
    }

    cout << "\nPlayer " << player
         << ", Enter position (1-9): ";
    cin >> choice;
```

Display Logic

- Infinite game loop using while(true) keeps the game running
- Nested loops traverse the 2D array efficiently
- Separators create visual grid structure
- Player prompt requests position input

The display refreshes after each move, showing the current game state clearly to both players.

Part 3: Move Validation

```
int row = (choice - 1) / 3;
int col = (choice - 1) % 3;

if (choice < 1 || choice > 9 ||
    board[row][col] == 'X' ||
    board[row][col] == 'O') {
    cout << "Invalid move! Try again.\n";
    continue;
}

board[row][col] = player;
moves++;
```

Validation Steps

1

Position Conversion

Translate user input (1-9) to array indices using division and modulo

2

Range Check

Ensure input falls within valid 1-9 range

3

Occupied Check

Prevent overwriting existing X or O marks

4

Update Board

Place player's mark and increment move counter

Part 4: Winning Condition Logic

```
// Check rows and columns in the board
for (int i = 0; i < 3; i++) {
   if ((board[i][0] == player && board[i][1] == player && board[i][2] == player) ||
      (board[0][i] == player && board[1][i] == player && board[2][i] == player))
      {
      cout << "\n Hurrey Player " << player << " wins!\n";
      return 0;
   }
 }

// Check diagonals in the board
 if ((board[0][0] == player && board[1][1] == player && board[2][2] == player) ||
    (board[0][2] == player && board[1][1] == player && board[2][0] == player))
    {
    cout << "\n Hurrey Player " << player << " wins!\n";
    return 0;
 }
```

Win Detection Strategy

Row Check

Iterates through each row to determine if three identical player marks are present.

Column Check

Scans every column to identify three consecutive identical player marks.

Diagonal Check

Examines both main diagonals for a sequence of three identical player marks.

Draw Detection & Player Switching

End Game Logic

```
if (moves == 9) {
    cout << "It's a draw!";
    return 0;
}

player = (player == 'X') ? 'O' : 'X';
```

After each validated move, the program checks if all 9 positions are filled. If no winner exists at that point, the game declares a draw.

The ternary operator elegantly switches between players: if the current player is 'X', it becomes 'O', and vice versa. This ensures proper turn alternation throughout the game.

Key Programming Features

2D Arrays

Matrix-based board representation for efficient position tracking and access

Loop Structures

while and for loops control game flow, board display, and win condition checking

Conditionals

if-else statements validate moves and determine game outcomes dynamically

Ternary Operator

Compact player-switching logic using the ? : syntax for cleaner code

Game Logic

Complete turn-based system with validation, win detection, and draw handling

Project Summary

Complete Implementation

Fully functional console-based Tic-Tac-Toe game demonstrating core C++ concepts

Learning Outcomes

Hands-on experience with arrays, loops, conditionals, and game logic design

Future Enhancements

AI opponent, GUI interface, score tracking, and difficulty levels can be added

This project successfully demonstrates fundamental programming principles while creating an engaging, interactive application. The modular structure makes it easy to extend and enhance with additional features.