THE TIC TAC TOE

Course: Introduction to problem solving and programming by G Ganeshan sir
Submitted By: Vanshika Shankar
Reg No: 25BHI10135
Date: November 25'25
Technology Stack: Python, Pygame

INTRODUCTION:

This project implements a classic tic tac toe game with the help of python and pygame library. This game provides a graphical platform where two players can play against each other locally. The implementation focuses on clean code architecture and proper game state management.

The game follows traditional Tic Tac Toe rules where players alternate placing X's and O's on a 3x3 grid, with the aim of getting three marks in a row horizontally, vertically, or diagonally. The system automatically detects win conditions and tie games, providing visual feedback and allowing players to restart.

PROBLEM STATEMENT:

The traditional form of this game which is played on paper has several limitations:

- Requires physical materials (paper and pencil)
- No automatic win detection
- Limited visual appeal and engagement
- Difficulty in tracking the game history
- No immediate restart capability

Objective: develop a digital Tic Tac Toe game that provides instant win detection, attractive interface, seamless gaming experience and easy game restart functionality.

FUNCTIONAL REQUIREMENTS:

Game Board Display

- Display a 3x3 grid game board
- Clear visual separation between cells
- Responsive to different screen sizes

Player Input Handling

- Accept mouse clicks to place marks
- Prevent placing marks on occupied cells
- Alternate between Player X and Player O

Win Detection

- Check horizontal wins (3 rows)
- Check vertical wins (3 columns)
- Check diagonal wins (2 diagonals)
- Declare winner immediately

Tie Detection

- Detect when all cells are filled
- Declare tie game when no winner exists

Visual Feedback

- Display X and O symbols clearly
- Show game over screen with winner
- Provide visual indication of game state

Game Reset

- Allow players to restart game
- Clear board and reset to initial state
- Keyboard shortcut for quick restart

NON-FUNCTIONAL REQUIREMENTS:

Performance

- Game should respond to clicks within 50ms
- No lag or delay in symbol drawing

Usability

- Intuitive interface requiring no instructions
- Clear visual distinction between X and O

- Colour scheme should be pleasant and not cause eye strain

Reliability

- Prevent crashes from unexpected inputs
- Graceful window closure handling

Maintainability

- Well-commented code for future modifications
- Modular function design

Portability

- Run on Windows, macOS, and Linux
- Require only Python and Pygame
- No platform-specific dependencies

SYSTEM ARCHITECTURE:

Event driven architecture - The game responds to user inputs (mouse clicks, keyboard presses) and updates the display accordingly.

Data Flow: User Input → Input Handler → Game Logic → State Update → Display Layer → Screen

DESIGN DIAGRAMS:

Use Cases:

1. Start Game
2. Place Mark (X or O)
3. View Game Board
4. Check Winner
5. Restart Game
6. Exit Game

Relationships:

- Player initiates all use cases
- "Place Mark" includes "Check Winner"
- "Check Winner" may trigger "Display Winner"

Workflow Diagram:

START

↓

Initialize Pygame & Display Board

↓

Wait for Player Click

↓

Is Cell Empty? → NO → Wait for Another Click

↓ YES

Place Current Player's Mark

↓

Check for Winner

↓

Winner Found? → YES → Display Winner → Wait for Restart

↓ NO

Board Full? → **YES** → Display Tie → Wait for Restart

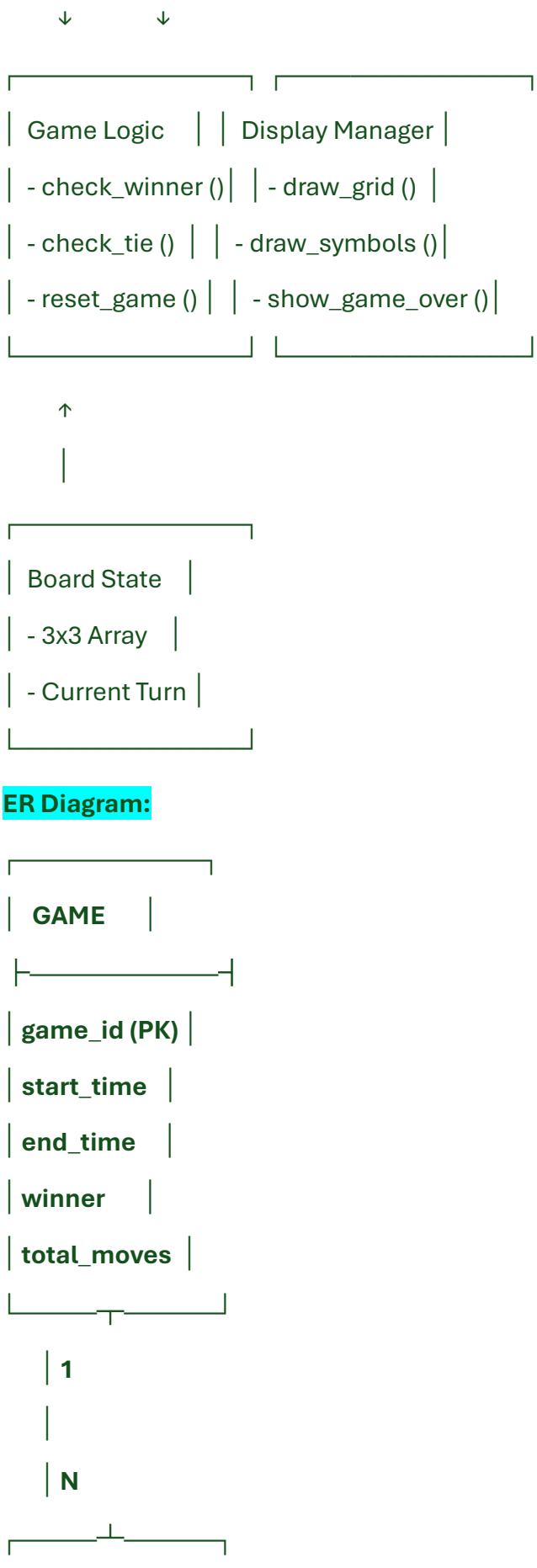
↓ NO

Switch Player Turn

↓

(Loop back to Wait for Player Click)

Component Diagram:


Game Logic
- check_winner ()
- check_tie ()
- reset_game ()
Display Manager
- draw_grid ()
- draw_symbols ()
- show_game_over ()
Board State
- 3x3 Array
- Current Turn

ER Diagram:

GAME
game_id (PK)
start_time
end_time
winner
total_moves
1
N

DESIGN DECISIONS AND RATIONALE:

Colour Scheme

Decision: Teal background with contrasting colours for X and O
Rationale:

- Reduces eye strain during extended play
- Clear visual distinction between elements
- Modern, appealing aesthetic

Grid-Based Coordinate System

Decision: 3x3 array to represent board state
Rationale:

- Intuitive mapping to visual grid
- Easy to implement win detection algorithms
- Simple to reset and manipulate

Turn-Based Boolean Flag

Decision: Single Boolean variable (x_turn) instead of player object
Rationale:

- Minimal state management for two-player game
- Fast turn switching
- Reduces code complexity for simple use case

IMPLEMENTATION DETAILS:

Board Representation:

board = [['', '', ''] , ['', '', ''] , ['', '', '']]

- 2D list structure for O (1) access time
- Empty strings represent unoccupied cells
- 'X' and 'O' strings represent player marks

Mouse Click to Grid Conversion:

clicked_row = mouse_y // 200

clicked_col = mouse_x // 200

Win Detection Algorithm:

- Time Complexity: O (1) - checks fixed number of conditions
- Checks 8 possible win conditions (3 rows + 3 columns + 2 diagonals)
- Returns winner immediately when found

SCREENSHOTS/ RESULTS:

Main Game Screen: [Describe: Clean 3x3 grid with teal background and dark teal lines]

Mid Game State: [Describe: Board showing several X's and O' s placed, demonstrating gameplay]

Win Condition Display: [Describe: Semi-transparent overlay showing "X Wins!" with restart instruction]

Tie Game Display: [Describe: Full board with "Tie Game!" message]

TESTING APPROACH:

Functional Testing

- Win detection - all 8-win conditions
- Tie detection when board is full
- Turn alternation between players
- Game reset functionality

Boundary Testing

- Clicking outside grid boundaries

- Clicking exactly on grid lines
- Multiple rapid clicks on same cell

Usability Testing

- Visual clarity of X and O symbols
- Game over screen readability
- Restart process intuitiveness

Performance Testing

- Response time to clicks
- Memory usage

All test cases passed successfully. No critical bugs identified. Minor enhancements noted for future versions.

CHALLENGES FACED:

- Converting mouse pixel coordinates to grid positions accurately.
- Preventing further moves after game ends.
- Drawing clean X and O symbols that scale properly.
- Properly resetting all game state variables.

LEARNINGS AND KEY TAKEAWAYS:

- **Pygame fundamentals** - Display initialization, rendering loops, and event handling.
- **Problem decomposition** - Breaking complex game logic into manageable functions.
- **Code organization** - Importance of clear structure and documentation.
- Separate presentation logic from game logic.
- Use descriptive variable and function names.
- Add comments for complex logic sections.
- Test edge cases early in development.

FUTURE ENHANCEMENTS:

- Smooth transitions when placing X and O.
- Display win count for each player across multiple games.
- Allow players to undo their last move.
- Statistics on win rates, common patterns.

REFERENCES:

- Pygame Official Documentation - https://www.pygame.org/docs/
- Python Official Documentation - https://docs.python.org/3/
- "Making Games with Python & Pygame" by Al Sweigart
- Pygame Tutorials - pygame.org/wiki/tutorials