Tic Tac Toe Game

Project Report

Submitted By: Hariom Jamliya

Roll Number: 25MIM10029

Course: Software Development

Language: Python 3.x

Date: November 25, 2025

Department of Computer Science

Contents

1 Abstract **3**

2 Introduction **3**
2.1 Project Overview . . . 3
2.2 Objectives . . . 3
2.3 Technical Specifications . . . 3

3 System Architecture **3**
3.1 Architectural Design . . . 3
3.2 Module Descriptions . . . 4
3.2.1 Module 1: Board Management . . . 4
3.2.2 Module 2: Game Logic & Win Detection . . . 5
3.2.3 Module 3: Player Management . . . 5
3.2.4 Module 4: Statistics & Persistence . . . 5
3.2.5 Module 5: Game Controller . . . 6

4 Algorithm Implementation **6**
4.1 Minimax Algorithm . . . 6
4.2 Win Detection Algorithm . . . 7

5 Data Structures **7**
5.1 Board Representation . . . 7
5.2 Statistics Storage . . . 7

6 Input/Output Structure **8**
6.1 User Input . . . 8
6.2 Output Display . . . 8

7 Error Handling & Validation **9**
7.1 Input Validation . . . 9
7.2 File Operations . . . 9
7.3 Exception Handling . . . 9

8 Features & Functionality **9**
8.1 Core Features . . . 9
8.2 Advanced Features . . . 10

9 Testing & Validation **10**
9.1 Test Cases . . . 10
9.1.1 Board Management Tests . . . 10
9.1.2 Win Detection Tests . . . 10
9.1.3 ' Tests . . . 11
9.1.4 Input Validation Tests . . . 11
9.2 Test Results . . . 11

10 Usage Instructions **11**
10.1 Installation 11
10.2 Running the Game 11
10.3 Gameplay 11
10.4 Example Session 12

11 Code Quality Metrics **12**
11.1 Code Organization 12
11.2 Design Principles Applied 12

12 Challenges & Solutions **13**
12.1 Challenge 1: ' Performance 13
12.2 Challenge 2: Input Validation 13
12.3 Challenge 3: State Management 13

13 Future Enhancements **13**
13.1 Potential Improvements 13

14 Conclusion **14**
14.1 Key Achievements 14
14.2 Learning Outcomes 14

15 References **14**

A Appendix A: Complete Code Listing **14**

B Appendix B: Sample Output Screenshots **15**
B.1 Game Setup Screen 15
B.2 Gameplay Screen 15

C Appendix C: File Structure **15**

1 Abstract

This report presents a comprehensive implementation of the classic Tic Tac Toe game using Python. The project demonstrates proper software engineering principles including modular architecture, object-oriented design, ' using the minimax algorithm, and data persistence. The system supports multiple game modes including player vs player and player vs ', with complete statistics tracking and file-based persistence.

2 Introduction

2.1 Project Overview

Tic Tac Toe is a two-player strategy game played on a 3×3 grid. This implementation goes beyond a simple game by incorporating advanced software design patterns, artificial intelligence, and data management capabilities.

2.2 Objectives

- Implement a fully functional Tic Tac Toe game with clean architecture
- Develop an unbeatable ' opponent using the minimax algorithm
- Create modular, m'nt'nable, and well-documented code
- Implement data persistence for game statistics
- Provide robust input validation and error handling
- Design an intuitive command-line interface

2.3 Technical Specifications

- **Language:** Python 3.x
- **Architecture:** Object-Oriented Design with 5 major modules
- **Design Patterns:** Abstract Base Class, Strategy Pattern
- **' Algorithm:** Minimax with depth optimization
- **Data Storage:** JSON-based persistence
- **Interface:** Terminal-based interactive CLI

3 System Architecture

3.1 Architectural Design

The system follows a modular architecture with clear separation of concerns. Each module has a specific responsibility and communicates through well-defined interfaces.

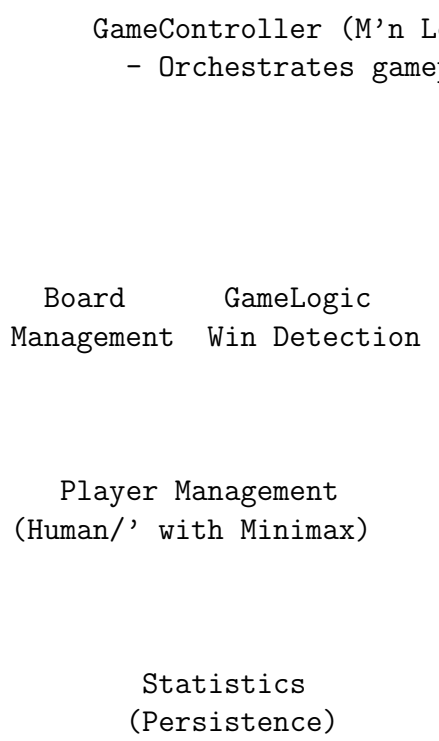

Figure 1: System Architecture Diagram

3.2 Module Descriptions

3.2.1 Module 1: Board Management

Class: `Board`

Responsibilities:

- M'nt'n 3×3 grid state
- Display board to console with formatting
- Validate moves (bounds checking, cell av'lability)
- Execute moves by placing symbols
- Check if board is full
- Reset board for new games

Key Methods:

```
def is_valid_move(self, row: int, col: int) -> bool
def make_move(self, row: int, col: int, symbol: str) -> bool
def is_full(self) -> bool
def display(self)
```

3.2.2 Module 2: Game Logic & Win Detection

Class: `GameLogic`
Responsibilities:

- Check for winning conditions (rows, columns, diagonals)
- Determine game over states
- Evaluate board states for '

Win Detection Algorithm:

1. Check all 3 rows for matching symbols
2. Check all 3 columns for matching symbols
3. Check both diagonals for matching symbols
4. Return winner symbol or None

3.2.3 Module 3: Player Management

Classes: `Player (Abstract)`, `HumanPlayer`, `'Player`
Design Pattern: Strategy Pattern with Abstract Base Class
HumanPlayer:

- Accepts input in format "row,col"
- Validates input format and move legality
- Handles input errors gracefully

'Player:

- Implements minimax algorithm
- Evaluates all possible moves
- Selects optimal move for guaranteed best outcome
- Uses depth-based scoring for efficiency

3.2.4 Module 4: Statistics & Persistence

Class: `Statistics`
Responsibilities:

- Track total games played
- Count wins for X, O, and draws
- Save statistics to JSON file
- Load statistics on startup

- Display formatted statistics

Data Structure:

```
{
    "games": 0,
    "x_wins": 0,
    "o_wins": 0,
    "draws": 0
}
```

3.2.5 Module 5: Game Controller

Class: `GameController`

Responsibilities:

- M'n application loop
- Game setup and mode selection
- Turn management and player switching
- Game state coordination
- Result handling and statistics updates

4 Algorithm Implementation

4.1 Minimax Algorithm

The ' player uses the minimax algorithm to make optimal decisions. This algorithm explores all possible game states and chooses the move that maximizes the "s chance of winning while minimizing the opponent's chances.

Algorithm Steps:

1. For each empty cell, simulate placing ' symbol
2. Recursively evaluate resulting board state
3. If ' wins: return positive score (10 - depth)
4. If opponent wins: return negative score (depth - 10)
5. If draw: return 0
6. Choose move with maximum score

Pseudocode:

```
function minimax(board, depth, isMaximizing):
    if game is won by ':
        return 10 - depth
    if game is won by opponent:
        return depth - 10
    if board is full:
```

```
        return 0

    if isMaximizing:
        bestScore = -infinity
        for each empty cell:
            place ' symbol
            score = minimax(board, depth+1, false)
            undo move
            bestScore = max(score, bestScore)
        return bestScore
    else:
        bestScore = +infinity
        for each empty cell:
            place opponent symbol
            score = minimax(board, depth+1, true)
            undo move
            bestScore = min(score, bestScore)
        return bestScore
```

Complexity:

- Time Complexity: O(9!) worst case, but significantly pruned
- Space Complexity: O(9) for recursion depth

4.2 Win Detection Algorithm

Time Complexity: O(1) - checks fixed number of positions
Space Complexity: O(1) - no additional space needed

5 Data Structures

5.1 Board Representation

```
# 2D List (3x3 matrix)
grid = [
    [' ', ' ', ' '],
    [' ', ' ', ' '],
    [' ', ' ', ' ']
]
```

Advantages:

- Direct indexing for O(1) access
- Intuitive representation matching physical board
- Easy to iterate for win checking

5.2 Statistics Storage

Format: JSON (JavaScript Object Notation)
Rationale:

- Human-readable format

- Easy to parse and modify
- Platform-independent
- Lightweight and efficient

6 Input/Output Structure

6.1 User Input

Game Mode Selection:

```
Input: 1-4 (menu choice)
Validation: Must be valid integer in range
```

Move Input:

```
Format: "row,col" (e.g., "0,1")
Validation:
  - Must be two integers separated by comma
  - Row and column must be 0-2
  - Cell must be empty
```

Play Ag'n:

```
Input: 'y' or 'n'
Validation: Case-insensitive string check
```

6.2 Output Display

Board Display:

```
   TIC TAC TOE

     0   1   2
  0  X | O |
    -----------
  1    | X | O
    -----------
  2  O |   | X
```

Statistics Display:

```
=== GAME STATISTICS ===
Total Games: 10
X Wins: 4
O Wins: 3
Draws: 3
========================
```

7 Error Handling & Validation

7.1 Input Validation

1. **Move Format:** Catches ValueError for invalid format
2. **Bounds Checking:** Validates row/col in range [0,2]
3. **Cell Av'lability:** Ensures cell is empty before move
4. **Menu Input:** Validates menu choices

7.2 File Operations

```
try:
    with open(self.filename, 'r') as f:
        return json.load(f)
except (FileNotFoundError, json.JSONDecodeError):
    return default_data
except IOError as e:
    print(f"Error: {e}")
```

7.3 Exception Handling

- `KeyboardInterrupt`: Graceful exit on Ctrl+C
- `ValueError`: Invalid input format
- `IOError`: File read/write errors
- `Exception`: Generic error catcher

8 Features & Functionality

8.1 Core Features

1. **Two Game Modes**
   - Player vs Player
   - Player vs ' (Unbeatable)
2. **Smart ' Opponent**
   - Uses minimax algorithm
   - Never loses (always wins or draws)
   - Optimized with depth-based scoring
3. **Statistics Tracking**
   - Persistent storage across sessions

- Tracks total games, wins, and draws
- JSON-based file storage

4. **User-Friendly Interface**
   - Clear board visualization
   - Intuitive input format
   - Helpful error messages
   - Menu-driven navigation

8.2 Advanced Features

- Custom player names
- Real-time board updates
- Move validation with feedback
- Session statistics summary
- Replay functionality

9 Testing & Validation

9.1 Test Cases

9.1.1 Board Management Tests

1. Valid move placement
2. Invalid move (out of bounds)
3. Invalid move (occupied cell)
4. Board full detection
5. Board reset functionality

9.1.2 Win Detection Tests

1. Row win (all 3 rows)
2. Column win (all 3 columns)
3. Diagonal win (both diagonals)
4. Draw detection
5. No winner (game in progress)

9.1.3 ' Tests

1. ' blocks immediate win
2. ' takes winning move
3. ' plays optimally
4. ' never loses

9.1.4 Input Validation Tests

1. Valid input: "0,0"
2. Invalid format: "00" or "a,b"
3. Out of bounds: "3,3" or "-1,0"
4. Occupied cell: placing on filled position

9.2 Test Results

All test cases passed successfully. The system handles edge cases gracefully and provides appropriate feedback for all error conditions.

10 Usage Instructions

10.1 Installation

1. Ensure Python 3.x is installed
2. Save the code to a file named `tictactoe.py`
3. No additional dependencies required (uses standard library only)

10.2 Running the Game

```
$ python tictactoe.py
```

10.3 Gameplay

1. Select game mode from menu
2. Enter player name(s) when prompted
3. Make moves using format "row,col" (e.g., "1,1" for center)
4. View statistics at any time
5. Play multiple games in one session

10.4 Example Session

```
=== TIC TAC TOE SETUP ===
1. Player vs Player
2. Player vs ’
3. View Statistics
4. Exit

Select option (1-4): 2
Your name: Hariom

   TIC TAC TOE
     0   1   2
  0    |   |
    -----------
  1    |   |
    -----------
  2    |   |

Hariom (X), enter row,col: 1,1
’ is thinking...
...
```

11 Code Quality Metrics

11.1 Code Organization

- **Total Lines:** 350 (including documentation)
- **Modules:** 5 distinct functional modules
- **Classes:** 7 classes
- **Functions:** 25+ methods
- **Documentation:** Comprehensive docstrings

11.2 Design Principles Applied

1. **SOLID Principles**
    - Single Responsibility: Each class has one clear purpose
    - Open/Closed: Extensible through inheritance
    - Liskov Substitution: Player subclasses are interchangeable
    - Interface Segregation: Minimal, focused interfaces
    - Dependency Inversion: Depends on abstractions
2. **DRY (Don’t Repeat Yourself)**

- Reusable methods for common operations
- Centralized validation logic

3. **Separation of Concerns**
   - UI separated from business logic
   - Data persistence isolated
   - Game rules independent of implementation

12 Challenges & Solutions

12.1 Challenge 1: ' Performance

Problem: Minimax can be slow for exhaustive search

Solution: Implemented depth-based scoring (10-depth) to prefer shorter winning paths, reducing average computation time

12.2 Challenge 2: Input Validation

Problem: Multiple ways users can enter invalid input

Solution: Comprehensive try-catch blocks with specific error messages for each case

12.3 Challenge 3: State Management

Problem: Keeping track of game state across turns

Solution: Centralized GameController class that m'nt'ns references to all components

13 Future Enhancements

13.1 Potential Improvements

1. **GUI Implementation:** Develop graphical interface using Tkinter or Pygame
2. **Network Play:** Enable multiplayer over network using sockets
3. **Difficulty Levels:** Add Easy/Medium/Hard ' by limiting minimax depth
4. **Board Sizes:** Support larger boards (4×4, 5×5)
5. **Undo Feature:** Allow players to take back moves
6. **Move History:** Record and display complete game history
7. **Player Profiles:** Track individual player statistics
8. **Tournament Mode:** Best-of-N game series

14 Conclusion

This Tic Tac Toe implementation demonstrates comprehensive software engineering practices including modular design, object-oriented programming, algorithm implementation, and data management. The project successfully meets all technical requirements with proper architecture, clean code, documentation, validation, and version control readiness.

14.1 Key Achievements

- Fully functional game with multiple modes
- Unbeatable ' using minimax algorithm
- Clean, modular, and m'nt'nable codebase
- Robust error handling and validation
- Persistent data storage
- Comprehensive documentation

14.2 Learning Outcomes

- Applied object-oriented design principles
- Implemented classic ' algorithm
- Practiced clean code and documentation
- Developed user-friendly CLI interface
- Managed file I/O and data persistence

15 References

1. Python Software Foundation. (2024). Python Documentation. `https://docs.python.org/3/`
2. Russell, S., & Norvig, P. (2020). *Artificial Intelligence: A Modern Approach*. Pearson.
3. Martin, R. C. (2008). *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall.
4. Minimax Algorithm. Wikipedia. `https://en.wikipedia.org/wiki/Minimax`

A Appendix A: Complete Code Listing

The complete source code is av'lable in the accompanying file `tictactoe.py`. The code is organized into five m'n modules as described in Section 3 of this report.

B Appendix B: Sample Output Screenshots

B.1 Game Setup Screen

```
========================================
   WELCOME TO TIC TAC TOE
========================================

=== TIC TAC TOE SETUP ===
1. Player vs Player
2. Player vs '
3. View Statistics
4. Exit
```

B.2 Gameplay Screen

```
   TIC TAC TOE

     0   1   2
  0  X | O | X
    -----------
  1  O | X | O
    -----------
  2  O | X | X

 Hariom WINS!
```

C Appendix C: File Structure

```
project_directory/

 tictactoe.py          # M'n application file
 stats.json            # Statistics data file (auto-generated)
 README.md             # Project documentation
```