Cairo University

Faculty of Engineering

Electronics and Electrical Communications Engineering

TIC TAC TOE

SRS DOCUMENTATION

Supervised by: Dr. **Omar Nasr**

Table of Contents

Advanced Tic Tac Toe Game

Version: 3.0
Date: June 20, 2025
Prepared by: Development Team
Project Supervisor: Dr. Omar Nasr

---

1. Introduction

1.1 Purpose

This SRS document describes the functional and non-functional requirements for the Advanced Tic Tac Toe Game project. This document serves as a reference for the developers, testers, and stakeholders to establish a clear understanding of the system requirements and expected behavior.

1.2 Intended Audience

- Development team members
- Project supervisor (Dr. Omar Nasr)
- Quality assurance testers
- End users (players)

1.3 Product Scope

This project implements an advanced version of the Tic Tac Toe game in C++ with features including:

- Player vs Player and Player vs AI game modes with multiple difficulty levels.
- Intelligent AI using the Minimax algorithm
- User registration and login with password hashing
- Personalized game history tracking and replay functionality.

- Graphical User Interface using Qt
- Unit and integration testing with Google Test
- Continuous Integration and Deployment via GitHub Actions
- Secure data storage via SQLite

1.4 Definitions and Acronyms

Term	**Description**
SRS	Software Requirements Specification
AI	Artificial Intelligence
PvP	Player vs Player
PvAI	Player vs AI
GUI	Graphical User Interface
SQLite	Lightweight database system
Qt	Cross-platform application framework

1.5 References

- IEEE 830-1998 SRS Standard
- Qt Documentation
- SQLite Documentation
- Google Test Framework

2. Overall Description

2.1 Product Overview

The Advanced Tic Tac Toe Game is a desktop application developed in C++ using the Qt framework, providing a modern and interactive graphical interface with animated elements for engaging user experience. The system features an intelligent AI opponent powered by strategic algorithms (e.g., Minimax with alpha-beta pruning), and supports both player-vs-player and player-vs-AI modes. It includes a secure user authentication system with password hashing, allowing players to create profiles, log in, and track their personalized game statistics. The application maintains a persistent game history database using SQLite, enabling users to review and replay past games. It integrates Google Test for comprehensive unit and integration testing and uses GitHub for version control along with GitHub Actions to automate CI/CD processes, ensuring code quality and smooth deployment workflows.

2.2 User Classes and Characteristics

- **Registered Players**: Users with accounts who can track statistics and game history
- **Guest Players**: Temporary users for PvP mode only
- **System Administrator**: Manages user data and system maintenance

2.3 Operating Environment

- **Operating System**: Windows
- **Programming Language**: C++
- **IDE:** Qt Creator / VS Code
- **GUI Framework**: Qt
- **Database**: SQLite
- **Testing Framework**: Google Test
- **Version Control System**: Git with GitHub as the remote repository
- **CI/CD Tool:** GitHub Actions
- **Storage Requirements:** 30 MB of free disk

2.4 Design and Implementation Constraints

- Must use C++ with Object-Oriented Programming principles
- GUI must be implemented using Qt framework
- Database operations must use SQLite
- GitHub Actions for CI/CD
- Google Test framework for testing

3. System Features and Requirements

3.1 Functional Requirements

3.1.1 User Authentication System

FR-001: User Registration

- The system shall allow new users to create accounts with username, password, and optional email

- The system shall validate username uniqueness
- The system shall enforce password strength requirements (minimum 4 characters)
- The system shall hash passwords using SHA-256 algorithm before storage

FR-002: User Login

- The system shall authenticate users with username and password credentials
- The system shall maintain user sessions during gameplay
- The system shall allow users to log in and log out

FR-003: User Profile Management

- The system shall display user statistics including games won, lost, and tied
- The system shall calculate and display win rate percentage
- The system shall allow users to view their profile information

3.1.2 Game Logic and Mechanics

FR-004: Game Board Management

- The system shall provide a 3x3 grid for gameplay
- The system shall track cell states (empty, X, or O)
- The system shall prevent moves on occupied cells
- The system shall alternate turns between players

FR-005: Win/Tie Detection

- The system shall check for winning conditions after each move (rows, columns, diagonals)
- The system shall detect tie conditions when the board is full with no winner
- The system shall end the game and display results when win/tie is detected

FR-006: Game Mode Selection

- The system shall support Player vs Player (PvP) mode
- The system shall support Player vs AI (PvAI) mode
- The system shall allow players to select their preferred symbol (X or O)

3.1.3 AI Opponent System

FR-007: AI Difficulty Levels

- The system shall provide Easy AI using random move selection
- The system shall provide Medium AI with basic strategy (win/block moves)
- The system shall provide Hard AI using minimax algorithm with alpha-beta pruning

FR-008: AI Move Generation

- The system shall generate AI moves within reasonable response time (< 2 seconds)
- The system shall ensure AI moves are valid and legal

3.1.4 Game History and Replay

FR-009: Game Recording

- The system shall record all game moves with timestamps
- The system shall store game results (win/loss/tie) with opponent information
- The system shall associate game records with user accounts

FR-010: Game History Display

- The system shall display the last 5 games for each user
- The system shall show game details including opponent, result, and timestamp
- The system shall provide access to complete game history

FR-011: Game Replay Functionality

- The system shall allow users to replay past games move by move
- The system shall provide replay controls (play, pause, reset, speed adjustment)
- The system shall display move sequence with proper timing

3.1.5 User Interface Requirements

FR-012: Main Menu Navigation

- The system shall provide intuitive navigation between game modes
- The system shall display user information and statistics prominently
- The system shall offer easy access to game history and settings

FR-013: Game Interface

- The system shall display the current player's turn clearly
- The system shall show game status and results prominently

FR-014: Visual Design Elements

- The system shall implement animated background elements with flying X and O symbols
- The system shall use consistent dark neon color schemes and typography
- The system shall provide visual effects for game events (wins, moves)

3.2 Non-Functional Requirements

3.2.1 Performance Requirements

NFR-001: Response Time

- User interface interactions shall respond within 100ms
- AI move generation shall complete within 2 seconds for Hard difficulty
- Database operations shall complete within 500ms

NFR-002: Resource Utilization

- The system shall use less than 200MB of RAM during execution
- Database file size shall be optimized for efficient storage

3.2.2 Security Requirements

NFR-003: Data Protection

- User passwords shall be stored using SHA-256 hashing
- User session data shall be protected from unauthorized access
- Database files shall be stored in protected user directories

NFR-004: Input Validation

- All user inputs shall be validated for format and content
- SQL injection prevention measures shall be implemented using prepared statements
- Buffer overflow protection shall be ensured for all string operations

3.2.3 Usability Requirements

NFR-005: User Experience

- The interface shall be intuitive for users aged 8 and above
- Game controls shall be accessible via mouse clicks
- Visual feedback shall be provided for all user actions

NFR-006: Accessibility

- Text shall be readable with appropriate font sizes and contrast
- Interface elements shall be clearly distinguishable
- Error messages shall be clear and actionable

3.2.4 Reliability Requirements

NFR-007: System Stability

- The application shall handle unexpected inputs gracefully
- Game state shall be preserved during normal operation
- Database integrity shall be maintained across sessions

NFR-008: Error Handling

- The system shall provide meaningful error messages
- Recovery mechanisms shall be available for common failures
- Application crashes shall be minimized through proper exception handling

3.2.5 Maintainability Requirements

NFR-009: Code Quality

- Code shall follow Google C++ Style Guidelines
- Functions shall be well-documented with clear comments
- Modular design shall facilitate future enhancements

NFR-010: Testing Requirements

- Unit test coverage shall exceed **90%** across all critical modules
- Integration tests shall validate complete workflows including login, gameplay, and history tracking

- Automated testing shall be triggered through the CI/CD pipeline to ensure consistent test enforcement.

3.3 External Interface Requirements

3.3.1 User Interfaces

- **Login Screen**: Username/password input with registration option
- **Registration Screen**: User account creation with validation
- **Main Menu**: Game mode selection and user profile access
- **Game Setup**: Difficulty selection and symbol choice for PvAI mode
- **Player Names**: Name input for PvP mode
- **Game Board**: 3x3 interactive grid with status displays
- **Game History**: List view with replay options
- **Game Replay**: Move-by-move replay with controls

3.3.2 Software Interfaces

- **Qt Framework**: GUI rendering and event handling
- **SQLite Database**: User data and game history storage
- **Google Test** for testing
- **GitHub Actions** for CI/CD

3.3.3 Hardware Interfaces

- **Mouse Input**: Primary interaction method for game moves
- **Keyboard Input**: Text entry for login and registration

3.4 System Constraints

3.4.1 Technical Constraints

- Must be developed in C++ using Object-Oriented Programming
- GUI must use Qt framework for cross-platform compatibility
- Database operations limited to SQLite for portability
- Memory usage must remain reasonable for desktop applications

4. Appendices

4.1 Dependencies

- Qt framework installation and configuration
- SQLite library availability
- C++ compiler with C++11 or later support
- Operating system GUI support

4.2 Future Enhancements

- Network multiplayer functionality
- Advanced AI personalities
- Customizable themes and animations
- Statistics export functionality