Performance Specification for Tic-Tac-Toe Game

Submitted to: Dr. Omar Nasr

1. Introduction

1.1 Purpose

This Performance Specification document outlines the performance requirements, metrics, and testing strategies for the Tic-Tac-Toe desktop application developed using Qt and C++. It ensures the system meets the non-functional performance requirements specified in the Software Requirements Specification (SRS) and aligns with the design in the Software Design Specification (SDS). The document defines target performance metrics, implementation strategies to achieve them, and methods to verify performance, focusing on response times, AI computation, file I/O, and resource usage.

1.2 Scope

The Tic-Tac-Toe game is a standalone desktop application with the following performance-critical components:

- **GUI Responsiveness**: Handling user inputs (e.g., button clicks) in real-time.
- **AI Computation**: Calculating AI moves in Player vs. AI (PvAI) mode, especially in Hard difficulty.
- **File I/O**: Saving and loading user credentials and game history.
- **Resource Usage**: Managing CPU and memory consumption for smooth operation.

This document specifies performance targets, describes how the system achieves them, and outlines testing approaches to validate performance on standard desktop hardware.

1.3 Definitions, Acronyms, and Abbreviations

- **SRS**: Software Requirements Specification (artifact_id: 5c495bfa-f306-434c-bab0-db414690f1fb).
- **SDS**: Software Design Specification (artifact_id: f459b087-1238-494d-86cb-08995fba81e7).
- **PvAI**: Player vs. AI mode.

- **GUI**: Graphical User Interface.
- **Minimax**: Algorithm for AI move selection.
- **Alpha-Beta Pruning**: Optimization for minimax to reduce computation.
- **QMessageBox**: Qt component for alerts.
- **QTimer**: Qt component for timed events (e.g., replay delays).
- **CPU**: Central Processing Unit.
- **I/O**: Input/Output (file operations).

1.4 References

- **SRS**: SRS_TicTacToe_Full.md.
- **SDS**: SDS_TicTacToe.md.
- **Qt Documentation**: https://doc.qt.io/.
- **Source Files**: main.cpp, mainwindow.cpp, registerwindow.cpp, gamemodewindow.cpp, playernamewindow.cpp, aidifficultywindow.cpp, symbolwindow.cpp, gameboard.cpp, aiplayer.cpp, gamehistory.cpp, historywindow.cpp, settingswindow.cpp, test_gameboard.cpp.
- **Storage Files**: registered_users.txt, tictactoe_<username>_history.json.

2. Performance Requirements

2.1 GUI Responsiveness

- **PR1.1**: The system shall process user inputs (e.g., button clicks for game moves, login, navigation) within **0.5 seconds**.
  - **Rationale**: Ensures a smooth user experience with no noticeable delays.
  - **Components**: MainWindow, GameModeWindow, PlayerNameWindow, AIDifficultyWindow, SymbolWindow, GameBoard, SettingsWindow, historywindow.
- **PR1.2**: The system shall update GUI elements (e.g., status labels, score displays, replay grid) within **0.3 seconds** after each action.
  - **Rationale**: Provides immediate visual feedback to users.

- **Components**: GameBoard::updateStatus, GameBoard::updateScore, historywindow::replayNextMove.

2.2 AI Computation

- **PR2.1**: The AI shall compute moves in **Easy mode** (random selection) within **0.1 seconds**.
  - **Rationale**: Random moves require minimal computation.
  - **Component**: AIPlayer::makeMove (random selection logic).
- **PR2.2**: The AI shall compute moves in **Medium mode** (minimax with depth 2) within **0.5 seconds** on a standard CPU (e.g., Intel i5, 2.5 GHz).
  - **Rationale**: Balances strategic decision-making with responsiveness.
  - **Component**: AIPlayer::minimax.
- **PR2.3**: The AI shall compute moves in **Hard mode** (minimax with depth 9) within **1 second** on a standard CPU.
  - **Rationale**: Ensures complex AI decisions remain responsive despite full board exploration.
  - **Component**: AIPlayer::minimax with alpha-beta pruning.

2.3 File I/O Performance

- **PR3.1**: The system shall save a game session to tictactoe_<username>_history.json within **0.5 seconds**.
  - **Rationale**: Ensures quick history updates without interrupting gameplay.
  - **Component**: GameHistory::saveGame.
- **PR3.2**: The system shall load up to **1000 game sessions** from tictactoe_<username>_history.json within **2 seconds**.
  - **Rationale**: Supports efficient history display for users with extensive game records.
  - **Component**: GameHistory::loadGames.
- **PR3.3**: The system shall save user credentials to registered_users.txt within **0.5 seconds**.
  - **Rationale**: Ensures quick registration/login processing.

- **Component**: MainWindow::saveRegisteredUsers.

- **PR3.4**: The system shall load user credentials from registered_users.txt within **0.5 seconds**.
  - **Rationale**: Supports fast login for users.
  - **Component**: MainWindow::loadRegisteredUsers.

2.4 Resource Usage

- **PR4.1**: The system shall use less than **100 MB of memory** during normal operation.
  - **Rationale**: Ensures compatibility with standard desktop hardware (e.g., 4GB RAM).
  - **Components**: All classes, especially GameBoard (board state) and historywindow (game history).
- **PR4.2**: The system shall maintain CPU usage below **20%** on a standard CPU during gameplay, except during Hard AI computations (up to 50% briefly).
  - **Rationale**: Prevents performance degradation on typical hardware.
  - **Components**: AIPlayer (Hard mode), GameBoard (game logic).

3. Performance Implementation Strategies

3.1 GUI Responsiveness

- **Qt Event Loop**: The system uses Qt's event-driven architecture to handle button clicks and UI updates efficiently.
  - **Implementation**: Signal-slot connections (e.g., GameBoard::onButtonClicked, MainWindow::on_LoginButton_clicked) process inputs asynchronously.
  - **Files**: gameboard.cpp, mainwindow.cpp, gamemodewindow.cpp, playernamewindow.cpp, aidifficultywindow.cpp, symbolwindow.cpp, settingswindow.cpp, historywindow.cpp.
- **Lightweight Widgets**: Qt widgets (QPushButton, QLabel, QLineEdit) are optimized for fast rendering.
  - **Implementation**: UI layouts defined in .ui files via Qt Designer, loaded via ui->setupUi.
- **Real-Time Updates**: Status and score labels are updated immediately after moves using GameBoard::updateStatus and updateScore.

3.2 AI Computation Optimization

- **Random Moves (Easy Mode)**:
  - **Implementation**: Uses rand() to select from a QVector<QPair<int, int>> of empty cells, ensuring minimal computation (AIPlayer::makeMove).
  - **File**: aiplayer.cpp.
- **Minimax with Alpha-Beta Pruning**:
  - **Implementation**: Medium (depth 2) and Hard (depth 9) modes use AIPlayer::minimax with alpha-beta pruning to reduce the number of evaluated nodes.
  - Pruning skips branches where beta <= alpha, significantly reducing computation time in Hard mode.
  - Board evaluation (evaluateBoard) checks only rows, columns, and diagonals for efficiency.
  - **File**: aiplayer.cpp.
- **Depth Limits**:
  - Medium: Depth 2 limits exploration to early game states.
  - Hard: Depth 9 explores the full game tree but is optimized by pruning.
  - **File**: aiplayer.cpp.

3.3 File I/O Optimization

- **JSON for Game History**:
  - **Implementation**: Uses Qt's QJsonDocument, QJsonArray, and QJsonObject for efficient JSON serialization/deserialization (GameHistory::saveGame, loadGames).
  - Incremental updates append new sessions to the JSON array, minimizing file rewrites.
  - **File**: gamehistory.cpp.
- **Text File for Credentials**:

- **Implementation**: registered_users.txt uses a simple username:hashed_password\n format, read/written with QTextStream for speed (MainWindow::saveRegisteredUsers, loadRegisteredUsers).
- **File**: mainwindow.cpp.

- **Error Handling**: File access failures are logged via qDebug, avoiding performance bottlenecks from retries.
- **Files**: gamehistory.cpp, mainwindow.cpp.

3.4 Resource Management

- **Memory Efficiency**:
  - **Implementation**: Minimal use of dynamic data structures (e.g., int[3][3] for board, QVector for temporary AI board, QList for moves).
  - Qt's memory management (e.g., parent-child object hierarchy) ensures automatic cleanup of UI objects.
  - **Files**: gameboard.cpp, aiplayer.cpp, gamehistory.cpp.
- **CPU Efficiency**:
  - **Implementation**: AI computations are the primary CPU-intensive task, optimized by alpha-beta pruning.
  - GUI updates and file I/O use lightweight Qt APIs to minimize CPU load.
  - **Files**: aiplayer.cpp, gameboard.cpp, gamehistory.cpp.

4. Performance Testing Strategies

4.1 GUI Responsiveness Testing

- **Test Case PT1.1**: Measure time from button click to UI update.
  - **Method**: Use QTime to record timestamps before and after button clicks (e.g., GameBoard::onButtonClicked, MainWindow::on_LoginButton_clicked).
  - **Target**: < 0.5 seconds for input processing, < 0.3 seconds for UI updates.
  - **Tools**: Qt Test framework, custom timing code.

- **Files**: Extend test_gameboard.cpp with timing tests.

- **Test Case PT1.2**: Simulate rapid button clicks to ensure stability.
  - **Method**: Programmatically trigger multiple clicks in GameBoard and verify no crashes or delays.
  - **Target**: System remains responsive under 10 clicks/second.
  - **Tools**: Qt Test, QTest::mouseClick.

4.2 AI Computation Testing

- **Test Case PT2.1**: Measure AI move time in Easy mode.
  - **Method**: Time AIPlayer::makeMove for random moves using QTime.
  - **Target**: < 0.1 seconds.
  - **File**: Add test to test_gameboard.cpp.
- **Test Case PT2.2**: Measure AI move time in Medium mode.
  - **Method**: Time AIPlayer::minimax with depth 2 on various board states.
  - **Target**: < 0.5 seconds on Intel i5, 2.5 GHz.
  - **File**: Add test to test_gameboard.cpp.
- **Test Case PT2.3**: Measure AI move time in Hard mode.
  - **Method**: Time AIPlayer::minimax with depth 9 on various board states.
  - **Target**: < 1 second on Intel i5, 2.5 GHz.
  - **File**: Add test to test_gameboard.cpp.

4.3 File I/O Testing

- **Test Case PT3.1**: Measure game session save time.
  - **Method**: Time GameHistory::saveGame for a single session using QTime.
  - **Target**: < 0.5 seconds.
  - **File**: Extend test_gameboard.cpp.

- **Test Case PT3.2**: Measure game history load time.
  - **Method**: Time GameHistory::loadGames with a JSON file containing 1000 sessions.
  - **Target**: < 2 seconds.
  - **File**: Extend test_gameboard.cpp.
- **Test Case PT3.3**: Measure user credentials save/load time.
  - **Method**: Time MainWindow::saveRegisteredUsers and loadRegisteredUsers with 100 users.
  - **Target**: < 0.5 seconds each.
  - **File**: Create new test file (e.g., test_mainwindow.cpp).

4.4 Resource Usage Testing

- **Test Case PT4.1**: Measure memory usage.
  - **Method**: Use system tools (e.g., Windows Task Manager, Linux top) or Qt's QMemoryInfo to monitor memory during gameplay and history loading.
  - **Target**: < 100 MB.
  - **Files**: All components, focus on GameBoard, historywindow.
- **Test Case PT4.2**: Measure CPU usage.
  - **Method**: Monitor CPU usage during gameplay (PvP, PvAI Hard mode) using system tools.
  - **Target**: < 20% average, < 50% peak during Hard AI moves.
  - **Files**: aiplayer.cpp, gameboard.cpp.

5. Performance Constraints

5.1 Hardware Constraints

- **CPU**: Standard desktop CPU (e.g., Intel i5, 2.5 GHz) for AI computations.
- **Memory**: Minimum 4GB RAM to support Qt and game data.
- **Storage**: Local file system with read/write permissions for registered_users.txt and JSON history files.

5.2 Software Constraints

- **Qt Framework**: Requires Qt 5 or 6 for GUI and JSON handling.
- **C++ Compiler**: Must support C++11 or later for standard library features (e.g., QVector).
- **File System**: Limited to local storage, no database or cloud support.

5.3 Algorithmic Constraints

- **Minimax Depth**: Limited to 2 (Medium) and 9 (Hard) to balance performance and strategy.
- **Alpha-Beta Pruning**: Essential for Hard mode to meet 1-second move time.
- **Random Number Generation**: Uses rand() for Easy mode, which is lightweight but not cryptographically secure.

6. Assumptions

- The system runs on a standard desktop with at least 4GB RAM and an Intel i5 CPU (2.5 GHz).
- Qt and C++ compiler are properly installed and configured.
- File system permissions allow read/write access to registered_users.txt and JSON history files.
- Users do not generate excessive game history (e.g., >10,000 sessions), which could impact load times.
- Performance tests are conducted in a controlled environment without significant background processes.

7. Appendices

7.1 Performance Metrics Summary

Metric	Target	Component	Test Method
GUI Input Response	< 0.5s	All UI classes	QTime timing, Qt Test
GUI Update	< 0.3s	GameBoard, historywindow	QTime timing
AI Move (Easy)	< 0.1s	AIPlayer	QTime in test_gameboard
AI Move (Medium)	< 0.5s	AIPlayer	QTime in test_gameboard
AI Move (Hard)	< 1s	AIPlayer	QTime in test_gameboard
Save Game Session	< 0.5s	GameHistory	QTime in test_gameboard
Load 1000 Games	< 2s	GameHistory	QTime in test_gameboard
Save/Load Credentials	< 0.5s	MainWindow	QTime in new test
Memory Usage	< 100 MB	All components	System tools, QMemoryInfo
CPU Usage	< 20% avg, < 50% peak	AIPlayer, GameBoard	System tools

7.2 Sample Performance Test Code (for test_gameboard.cpp)

```
void TestGameBoard::testAIMovePerformance() {

    GameBoard board("", nullptr);

    board.setGameMode(GameBoard::PlayerVsAI);

    board.setAIDifficulty("Hard");

    board.setPlayerSymbol('X');

    QTime timer;

    timer.start();

    int row, col;
```

```
    QVector<QVector<char>> aiBoard(3, QVector<char>(3, ' '));
    board.aiPlayer->makeMove(aiBoard, row, col);
    int elapsed = timer.elapsed();
    QVERIFY(elapsed < 1000); // Hard mode < 1 second
}
```