Test Documentation

1. Testing Strategy and Environment

Testing Framework

The project utilizes the **Qt Test** framework; a lightweight and powerful unit testing solution integrated within the Qt ecosystem. Qt Test enables the creation of robust test suites for C++ applications, particularly those built with Qt.

Testing Approach

The current testing approach focuses primarily on **Unit Testing**, where individual components and functions of the game's backend logic are tested in isolation or with controlled dependencies. This ensures that each module performs as expected before integration.

Test Environment

- **Operating System:** Windows 11
- **Qt Test Library Version:** 6.9.0
- **Qt Version:** 6.9.0
- **Compiler:** GCC 13.1.0 (MinGW 64-bit toolchain)

2. Test Suites Overview and Results

The testing effort is organized into four main test suites, each targeting a specific core component of the application: **TestGame**, **TestGameHistoryModel**, **TestUser**, and **TestDBManager**,

Test Game

- **Purpose:** To validate the core game mechanics, board state management, win/draw conditions, and the Artificial Intelligence (AI) opponent's logic.
- **Key Areas Covered:**
  - Initialization of the game board.
  - Validation of player moves (valid, invalid, out-of-bounds).
  - Correct player switching.
  - Accurate detection of all winning conditions (horizontal, vertical,

diagonal).
    - Accurate detection of draw conditions.
    - Retrieval of individual cell values and the full board state string.
    - Configuration and behavior of AI difficulty levels (Easy, Hard, Perfect/Impossible).
    - Correct evaluation of board states for minimax algorithm.
    - Proper construction and traversal of the game tree for AI decision-making.
    - Minimax algorithm's ability to find optimal moves (win, block, draw).
- **Total Test Cases Executed:** 25
- **Results:**
    - **Passed:** 25
    - **Failed:** 0
    - **Skipped:** 0
    - **Blacklisted:** 0
    - **Total Time:** 257ms

TestGameHistoryModel

- **Purpose:** To verify the data model responsible for managing and presenting personalized game history.
- **Key Areas Covered:**
    - Initial state of the history model.
    - Loading game history from the database.
    - Adding new game records to history.
    - Correct mapping and retrieval of data for display roles in a table view.
- **Total Test Cases Executed:** 6
- **Results:**
    - **Passed:** 6
    - **Failed:** 0
    - **Skipped:** 0
    - **Blacklisted:** 0
    - **Total Time:** 49ms

Test User

- **Purpose:** To ensure secure user authentication and profile management functionalities.
- **Key Areas Covered:**
  - Initial login state of a new User object.
  - Successful user registration (signup).
  - Handling of duplicate username registration attempts.
  - Successful user login with correct credentials.
  - Failure of user login with invalid credentials.
  - User logout functionality.
  - Verification of user login status.
- **Total Test Cases Executed:** 8
- **Results:**
  - **Passed:** 8
  - **Failed:** 0
  - **Skipped:** 0
  - **Blacklisted:** 0
  - **Total Time:** 2233ms (Note: This includes database operations for signup/login, which can be slower than pure in-memory operations.)

TestDBManager

- **Purpose:** To validate the interactions with the SQLite database for storing and retrieving user credentials and game records.
- **Key Areas Covered:**
  - Correct initialization and singleton pattern of the database manager.
  - Adding new users to the database.
  - Checking user credentials for login.
  - Verifying if a username already exists.
  - Saving game records to the database.
  - Retrieving game records for a specific user.
- **Total Test Cases Executed:** 8
- **Results:**
  - **Passed:** 8
  - **Failed:** 0
  - **Skipped:** 0
  - **Blacklisted:** 0
  - **Total Time:** 26ms

TestPerformance

- **Purpose:** To measure the execution time, resource usage, and scalability of key game operations, particularly focusing on AI algorithms and game simulations. This serves as an integration test for the efficiency of interacting components.
- **Key Areas Covered:**
  - **AI Move Performance:** Benchmarking response times for AI moves across different difficulty levels (Easy, Medium, Hard, Impossible).
  - **Algorithm Efficiency:** Measuring the time taken for core AI algorithms like game tree building and minimax calculations under various board scenarios.
  - **Game Simulation:** Benchmarking the time taken to simulate full games.
  - **Resource Usage:** Basic measurements of memory consumption and checks for potential memory leaks during intensive operations.
  - **Stress & Scalability:** Evaluating performance under high load (many iterations, concurrent games) and how performance scales with increased complexity or simultaneous operations.
- **Total Test Cases Executed:** 16 (individual benchmarks)
- **Results:**
  - **Passed:** 20
  - **Failed:** 0
  - **Skipped:** 0
  - **Blacklisted:** 0
  - **Total Time:** 12097ms (~12 seconds)

3. Overall Test Summary

All automated unit tests for the core game logic, user management, and database persistence have **passed successfully**.

Test Suite	Total Tests	Passed	Failed	Total Time (ms)
TestGame	25	25	0	257
TestGameHistoryModel	6	6	0	49
TestUser	8	8	0	2797
TestDBManager	8	8	0	26
TestPerformance	20	20	0	12097