Testing Documentation

for

Advanced Tic Tac Toe Game

VERSION 1.0

Prepared by:

- Abdelrahman Mohamed
- Mohamed Hisham Mohamed
- Salah Eldin Hassen
- Shahd Gamal
- Shahd Hamad

Table of Contents

1. Introduction

The testing documentation for the Advanced Tic Tac Toe Game provides a comprehensive overview of the testing efforts conducted to ensure the functionality, reliability, and performance of the game. This section outlines the purpose of the testing documentation, the intended audience, and reading suggestions for a better understanding of the testing process.

1.1 Purpose

The purpose of this testing documentation is to:

- Provide a detailed description of the testing strategies, test cases, and coverage reports conducted for the Advanced Tic Tac Toe Game.
- Ensure the correctness, reliability, and performance of the game functionalities through systematic testing.
- Serve as a reference guide for developers, testers, and project stakeholders to assess the quality and readiness of the game for release.

1.2 Intended Audience and Reading Suggestions

The testing documentation is intended for developers, testers, and project stakeholders involved in the design, implementation, and assessment of the Advanced Tic Tac Toe Game. To effectively utilize the documentation, readers are recommended to understand testing objectives, familiarize themselves with test strategies, review detailed test cases for Game and Player classes, analyze coverage reports, and summarize key findings and recommendations for ensuring the game's quality.

2. Testing Strategies

2.1 Unit Testing

Unit testing is a fundamental testing strategy employed in the testing documentation for the Advanced Tic Tac Toe Game. It involves testing individual units or components of the software in isolation to ensure their correctness and functionality. In the context of the game, unit testing focuses on validating the Game class, Player class, AI algorithms, and other critical components.

Approach:

1. **Game Class Tests**: Unit tests are designed to verify the game logic, including board initialization, player moves, win/tie detection, and game outcomes.
2. **Player Class Tests**: Tests are conducted to validate player actions, such as making moves, managing profiles, and interacting with the game interface.
3. **AI Algorithm Tests**: Unit tests assess the behavior of the AI opponent, ensuring that it follows the minimax algorithm with alpha-beta pruning and offers different difficulty levels**.**

4. **Session Class Tests:** Unit tests validate the session management functionalities, including adding games, updating scores, and retrieving session data.
5. **Database Class Tests**: Unit tests ensure the correctness of database operations, including saving and retrieving game data, user profiles, and session information.

Tool: The Qt Test framework is utilized for writing and executing unit tests, providing a structured approach to validate the functionality of individual components.

2.2 Integration Testing

Integration testing is essential to validate the interaction and integration of different modules within the Advanced Tic Tac Toe Game. It focuses on ensuring that the components work together seamlessly and that data flows correctly between them.

Approach:

1. **Game Play Integration:** Testing the interaction between the Game class, Player class, and GUI components to ensure smooth gameplay.
2. **User Authentication Integration:** Tests are conducted to validate player actions, such as making moves, managing profiles, and interacting with the game interface.
3. **AI Integration:** Testing the integration of the AI opponent module with the game logic to provide strategic gameplay experiences.

Tool: Integration tests are performed using the Qt Test framework and GitHub action as **(.yaml)** file**.**

3. Test Cases

3.1 Game Class Tests

➢ **Test Case 1:** Verify Board Initialization
- **Purpose:** Ensure the game board is correctly initialized.
- **Procedure:**

  1. Create a new Game instance.
  2. Verify that all cells in the board are empty.

- **Expected Result:** All board cells should be initialized to ' '.

➢ **Test Case 2:** Validate Player Moves
- **Purpose:** Ensure that player moves are correctly registered on the board.
- **Procedure:**

  1. Create a new Game instance.
  2. Make a move as the player.
  3. Verify that the board reflects the player's move.

- **Expected Result:** The board should correctly show the player's move.

- **Test Case 3:** Detect Win Condition
  - **Purpose:** Verify win condition detection logic.
  - **Procedure:**

    1. Create a new Game instance.
    2. Simulate a winning sequence of moves for the player.
    3. Verify that the game state is set to 'w' (win).

  - **Expected Result:** The game state should be 'w'.

3.2 Player Class Tests

- **Test Case 1:** Add Session
  - **Purpose:** Ensure the game board is correctly initialized.
  - **Procedure:**

    1. Create a new Player instance.
    2. Add a Session instance.
    3. Verify that the session count is incremented.

  - **Expected Result:** The session count should be incremented by 1.

- **Test Case 2:** Make Move
  - **Purpose:** Ensure that the player can make a move in the game.
  - **Procedure:**

    1. Create a new player instance.
    2. Simulate a move in the game.
    3. Verify that the move is registered correctly.

  - **Expected Result:** The move should be registered in the game.

3.3 AI Algorithm Tests

- **Test Case 1:** Validate Minimax Algorithm
  - **Purpose:** Ensure that the minimax algorithm calculates the best move.
  - **Procedure:**

    1. Create a new Game instance.
    2. Simulate a game state.
    3. Verify that the AI selects the optimal move using the minimax algorithm.

  - **Expected Result:** The AI should select the optimal move.

- **Test Case 2:** Test Difficulty Levels
  - **Purpose:** Validate different difficulty levels of the AI.
  - **Procedure:**

    1. Create a new Game instance.
    2. Set different difficulty levels for the AI.

3. Verify the behavior of the AI at each difficulty level.

- **Expected Result:** The AI's performance should vary according to the set difficulty level.

3.4 Session Class Tests

➢ **Test Case 1:** Add Game
- **Purpose:** Ensure that adding a game updates the session's game count and score correctly.
- **Procedure:**

1. Create a new Session instance.
2. Add a Game instance with a win state.
3. Verify that the game count and score are updated correctly.

- **Expected Result:** The game count should increment by 1, and the score should reflect the win.

➢ **Test Case 2:** Get Games Count
- **Purpose:** Verify the getGamesCount function returns the correct game count.
- **Procedure:**

1. Create a new Session instance.
2. Add multiple Game instances.
3. Verify that getGamesCount returns the correct number of games.

- **Expected Result:** The function should return the correct game count.

➢ **Test Case 3:** Get Games Pointer
- **Purpose:** Ensure that getGamesPointer returns a valid pointer to the games vector.
- **Procedure:**

1. Create a new Session instance.
2. Add multiple Game instances.
3. Verify that getGamesPointer returns a pointer to the games vector.

- **Expected Result:** The pointer should correctly reference the games vector.

3.5 Database Class Tests

➢ **Test Case 1:** Save Game Data
- **Purpose:** Ensure that game data is saved correctly to the database.
- **Procedure:**

1. Create a new Database instance.
2. Save game data.
3. Verify that the data is stored correctly in the database.

- **Expected Result:** The data should be stored in the database as expected.

➢ **Test Case 2:** Retrieve User Profile

- **Purpose:** Ensure that user profile data can be retrieved correctly.
- **Procedure:**

  1. Create a new Database instance.
  2. Retrieve a user profile.
  3. Verify that the profile data matches the expected values.

- **Expected Result:** The profile data should match the expected values.

4. Coverage Reports

Coverage reports are a crucial part of the testing strategy for the Advanced Tic Tac Toe Game. They provide insights into how much of the code is covered by the test cases, helping identify untested parts of the code.

➢ **Purpose**:

-Measure the percentage of the codebase executed by the test suite.

-Identify untested code sections.

-Ensure comprehensive testing by highlighting areas needing additional tests.

➢ **Tools:**

-Qt Test Framework: For running unit and integration tests.

-Coverage Tools: Tools like gcov, lcov, or Qt Creator's built-in coverage analysis tools to generate reports.

4.2 Key Coverage Metrics

Statement Coverage:

- Measures the percentage of executable statements executed.

- Goal: High statement coverage to ensure most code statements are tested.

Branch Coverage:

- Measures the percentage of branches (decision points) executed.

- Goal: High branch coverage to ensure all possible paths are tested.

Function Coverage:

- Measures the percentage of functions called during test execution.

- Goal: High function coverage to ensure most functions are tested.

Path Coverage:

- Measures the percentage of possible execution paths traversed.
- Goal: High path coverage to ensure all potential paths are tested.

4.3 Coverage Analysis Process

Run Tests: Execute the full test suite using the Qt Test framework.

Generate Reports: Use coverage tools to generate coverage data and reports.

Analyze Data:

- Review reports to identify covered and uncovered code sections
- analyze metrics to evaluate test suite effectiveness.

Identify Gaps:

- Highlight low or no coverage areas.
- Identify critical untested components.

Improve Coverage:

- Write additional tests for uncovered areas.
- Enhance existing tests for better coverage.
- Re-run tests and regenerate reports to ensure improvements.

4.4 Coverage Report Summary

➢ Example Coverage Report Summary:

Metric	Value
Statement Coverage	85%
Branch Coverage	75%
Function Coverage	90%
Path Coverage	70%

5. Conclusion

The testing strategy employed in this project successfully identified and addressed potential issues, leading to a highly reliable and functional Advanced Tic Tac Toe Game. The use of unit and integration testing, complemented by detailed coverage analysis, ensured that all critical aspects of the application were thoroughly tested.

5.1 Future Work

While the current testing strategy has achieved significant coverage and reliability, future efforts should focus on:

- **Expanding Test Cases**: Continuously adding new test cases to cover edge scenarios and newly added features.
- **Automated Testing**: Implementing automated testing pipelines to streamline the testing process and ensure continuous validation.
- **Performance Testing**: Conducting performance testing to assess and improve the application's responsiveness and efficiency under various conditions.

In conclusion, the structured and comprehensive testing approach adopted in this project has significantly contributed to the development of a robust and reliable Advanced Tic Tac Toe Game, setting a strong precedent for future testing and development endeavors.

5.2 References

➢ Qt Documentation : Qt Documentation | Home
➢ Chapter 1: Writing a Unit Test: Chapter 1: Writing a Unit Test | Qt Test 6.7.2
➢ Qt Test Overview: Qt Test Overview | Qt Test 6.7.2