Faculty of Engineering
Department of Electronics and Electrical Communications
ELC2080 Project

Advanced Tic-Tac-Toe

Testing Document

Name	ID
Ziad Mostafa	9231055
Ziad Emad	9230401
Ziad Abdel Hafeez	9230399
Sandra Atef	9230428
Robir Tamer	9230378

Under the supervision of:

Dr. Omar Ahmed Nasr

Table of Contents

1 Document Release History

#	Version No.	Release Date	Prepared By	Reviewed By	Preparation Comments
1	V1.0	17/6/2024	Robir Tamer	Ziad Emad	Initial Publication

2 Authors, Reviewers and Stakeholders’

Client Stakeholders Register

#	Name	Role in the project	Influence	Contact	Email
1	Dr. Omar Ahmed Nasr	Program Manager	SPOC	01143941832	omaranasr.phone@gmail.com

3 Introduction

This document provides comprehensive test documentation for the Tic-Tac-Toe game application. It outlines the testing objectives, environment, approach, and detailed test cases for various components of the game.

4 Testing Plan

4.1 Objective

The core objective of testing the Tic Tac Toe game is to ensure it functions flawlessly. This involves two main aspects:

1. **Component Verification:** We need to confirm that each individual component of the game, like the game board logic or the AI opponent, operates correctly according to its design.
2. **Robustness and User Experience:** We'll test how the game handles unexpected situations (edge cases) gracefully, without crashing or producing unintended results. Additionally, we'll assess how smoothly the game plays from a user's perspective, identifying any potential issues that might hinder their enjoyment.

4.2 Test Environment

- **Hardware:** Development machines, testing servers (Qt)
- **Operating Systems:** Windows
- **Compiler:** GCC
- **Library:** Qt

4.3 Approach

The testing approach for the Tic-Tac-Toe game involves a combination of unit testing, AI testing, GUI testing, integration testing (including End-to-End and API testing), and user acceptance testing. Each approach focuses on different aspects of the application to ensure comprehensive coverage.

4.3.1 Unit testing

Unit tests focus on individual components in isolation, ensuring they function correctly according to their design. For this project, unit tests primarily cover core game functionalities such as initialization, move validation, win condition detection, and board state management.

4.3.2 AI Testing

AI testing validates the behavior of the AI player, ensuring it makes reasonable and strategic moves across different difficulty levels. It also considers edge cases such as the AI playing against itself or starting the game.

4.3.3 GUI Testing

GUI testing focuses on the graphical user interface (UI) interactions, verifying that buttons, menus, and other UI elements respond correctly. It also checks if the UI updates appropriately after each move and simulates user input to test various scenarios.

4.3.4 Integration Testing

Integration testing examines how individual components of the game collaborate to deliver overall functionality. This includes:

4.3.4.1 End-to-End Testing

End-to-end testing meticulously simulates complete game sessions, mirroring the user's experience from start to finish. This comprehensive approach encompasses a wide range of scenarios, including:

- **User Authentication:** If the game incorporates user accounts, this testing phase verifies the login or registration process, ensuring secure access and proper user management.
- **Game Initiation:** Tests involving starting a new game, confirming the game board is correctly initialized with players and other necessary elements.
- **Gameplay Flow:** This stage rigorously examines user interactions throughout the game. It assesses how user moves are communicated to the game logic component and how the updated game state is accurately reflected back on the user interface.

- **AI Interaction:** It verifies the AI's behaviour during gameplay. This includes ensuring the AI makes strategic moves, responds appropriately to user actions, and adheres to the defined game rules.
- **Game Termination:** Testing encompasses various game-ending scenarios, such as win conditions, ties, and potential game over states. This ensures the game logic accurately detects these situations and displays the appropriate results to the user.
- **Game History Management:** It verifies that user wins, losses, and game data are saved persistently and can be retrieved accurately for future reference.

By simulating these comprehensive user journeys, we can proactively identify and address potential integration issues. For instance, an end-to-end test might uncover a scenario where a user successfully wins a game, but the game history functionality fails to record this victory. This comprehensive testing approach helps to ensure a robust and cohesive user experience.

4.3.4.2 API Testing

API (Application Programming Interface) testing focuses on meticulously examining the communication and data exchange between various components within the Tic-Tac-Toe game. This testing stage is crucial because different parts of the system, like the graphical user interface (GUI) and the game logic, often interact through well-defined APIs. Here are some key areas to consider in API testing:

- **GUI-Game Logic Interaction:** This testing meticulously verifies how user actions on the GUI (e.g., clicking a button to make a move) translate into calls to the game logic component. It also assesses how updates from the game logic (e.g., changes in the game state) are accurately reflected on the GUI (e.g., displaying the updated board state). By rigorously testing these interactions, we can ensure seamless communication between the user interface and the game logic, preventing potential issues that might arise from communication breakdowns.
- **Data Consistency:** This testing stage meticulously examines data passed between components to ensure consistency and accuracy throughout the game. For example, API testing verifies that the player's move information sent from the GUI matches the data received and processed by the game logic. This ensures a unified and reliable data flow across the system, preventing potential discrepancies that could lead to unexpected behaviours.

By employing these rigorous integration testing approaches, we can foster a robust and well-functioning Tic-Tac-Toe game, ensuring a seamless user experience and preventing potential issues from arising due to communication breakdowns or data inconsistencies between different parts of the system.

4.3.5 User Acceptance testing

Acceptance testing in the Tic-Tac-Toe game plays a crucial role in verifying that the game functions as intended from a user's perspective, fostering a seamless and enjoyable gameplay experience. This phase delves deeper than the individual component testing (unit testing) and integration testing stages, simulating real-world user interactions and scenarios.

4.3.5.1 Gameplay Functionality:

- **Starting a New Game:** Verify that users can effortlessly initiate a new game session. This includes selecting game modes (single player vs. two player), adjusting difficulty levels (for AI opponents), and successfully starting the game with a clear and empty board.
- **Move Validation:** Test that the game enforces valid moves. Users should not be able to place symbols outside the board boundaries or in already occupied cells.
- **Turn Management:** Ensure turns alternate between the human player and the AI opponent correctly. The game should prevent invalid actions during the AI's turn.
- **Win Conditions:** Test that the game accurately detects all winning scenarios (horizontal, vertical, diagonal rows with three matching symbols). The game should appropriately display the winner, update the personalized game history (adding the winner, date, and potentially other relevant details), and offer options to start a new game or view the game history.
- **Tie Conditions:** Verify that the game recognizes a tie when all cells are filled without a winner emerging. The game should display a tie message, update the game history, and offer options for replaying or exiting.

4.3.5.2 User Interface (UI) and User Experience (UX):

- **Intuitive Interface:** Test if the game interface is clear and easy to understand for users of all experience levels. The board layout, buttons, and menus should be visually appealing and logically arranged.
- **Visual Feedback:** Verify that the game provides appropriate visual feedback to users. This might include highlighting valid move options on the board, displaying animations for symbol placement, indicating the current player's turn (human or AI), and showcasing the AI's chosen move with a clear animation or visual cue.
- **Responsiveness:** Test that the game interface responds promptly to user actions. Clicking buttons, selecting moves, and navigating menus should be smooth and immediate.

- **Accessibility:** Consider testing the game's accessibility for users with disabilities. This might involve ensuring proper colour contrast for visually impaired users or implementing keyboard shortcuts for those who might have difficulty using a mouse.

4.3.5.3 Single Player Mode (vs. AI):

- **AI Functionality:** Test the AI opponent's behavior at different difficulty levels. Verify that the AI makes strategic moves that become more challenging as the difficulty increases.
- **AI Responsiveness:** Ensure the AI responds promptly to user moves and doesn't suffer from excessive delays in making its own choices.
- **AI Move Variety:** Test if the AI utilizes a variety of strategies to keep gameplay engaging and prevent it from becoming predictable at higher difficulty levels.

4.3.5.4 Personalized Game History:

- **Game History Access:** Verify that users can easily access their personalized game history. This history should include details like players involved (human vs. AI opponent difficulty level), and the final outcome (win, loss, or tie) and the moves one-by-one.

4.3.5.5 Error Handling:

- **Unexpected User Input:** Test how the game handles unexpected user actions, such as clicking outside the game board or attempting invalid moves. The game should provide clear error messages and prevent the application from crashing.

By performing these acceptance tests, you can ensure the Tic-Tac-Toe game delivers a polished and engaging experience for users. This testing focuses on core gameplay functionality, a user-friendly interface, a strategic AI opponent, and the ability to track personal game history

5 Test cases

5.1 Game Algorithm

5.1.1 Game Initialization

Criteria	Details
Purpose	Verify that the game board is correctly initialized with all cells set to empty.
Test steps	1. Create a new TicTacToe game instance. 2. Iterate over each cell in the game board. 3. Assert that each cell is initialized to EMPTY_CELL.
Expected Outcome	All cells in the game board should be set to EMPTY_CELL.

5.1.2 Make movement

Criteria	Details
Purpose	Verify that a move can be made by a player and that the move is correctly recorded on the board.
Test steps	1. Create a new TicTacToe game instance. 2. Make a move for PLAYER_X at position (0, 0). 3. Assert that the board reflects the move made by PLAYER_X.
Expected Outcome	The cell at position (0, 0) should be set to PLAYER_X.

5.1.3 Check the Winner

Criteria	Details
Purpose	Verify that the game can correctly identify a winning condition for a player.
Test steps	1. Create a new TicTacToe game instance. 2. Make moves for PLAYER_X to form a winning line. 3. Assert that isWinner correctly identifies PLAYER_X as the winner.
Expected Outcome	isWinner should return true for PLAYER_X.

5.1.4 Check the board full

Criteria	Details
Purpose	Verify that the game can correctly identify when the board is full.
Test steps	1. Create a new TicTacToe game instance. 2. Make moves to fill the entire board. 3. Assert that isBoardFull correctly identifies that the board is full.
Expected Outcome	isBoardFull should return true when the board is completely filled.

5.1.5 Check the board clear

Criteria	Details
Purpose	Verify that the game board can be cleared, resetting all cells to empty.
Test steps	1. Create a new TicTacToe game instance. 2. Make a move for PLAYER_X. 3. Clear the board. 4. Iterate over each cell in the game board. 5. Assert that each cell is set to EMPTY_CELL.
Expected Outcome	All cells in the game board should be reset to EMPTY_CELL.

5.1.6 Find the best move

Criteria	Details
Purpose	Verify that the game can find the best move for the AI player.
Test steps	1. Create a new TicTacToe game instance. 2. Make initial moves for PLAYER_X and PLAYER_O. 3. Find the best move for PLAYER_X. 4. Assert that the best move is within valid board coordinates.
Expected Outcome	The best move should be within the valid range of the board.

5.2 Login Page

These tests focus on the authentication and login functionalities of the application.

5.2.1 Test the input credentials

Criteria	Details
Purpose	Verify that the Authentication class correctly validates user credentials, including handling various input types and edge cases.
Test steps	1. Attempt to register and log in with valid username and password combinations. 2. Attempt to log in with incorrect passwords. 3. Attempt to log in with empty usernames or passwords. 4. Attempt to log in with usernames or passwords containing special characters. 5. Attempt to log in with very long usernames or passwords.
Expected Outcome	1. Valid credentials should allow successful registration and login. 2. Incorrect passwords should result in login failure. 3. Empty credentials should result in login failure. 4. Special characters and long credentials should be handled correctly without errors.

5.2.2 Password Hashing and Varification

Criteria	Details
Purpose	Verify that passwords are securely hashed and can be correctly verified.
Test steps	1. H ash a sample password. 2. Verify the hashed password against the original password. 3. Attempt to verify with a wrong password.
Expected Outcome	1. Hashing should produce different hashes for the same password due to salting. 2. Verification should succeed for correct passwords and fail for incorrect ones

5.2.3 Salt Generation

Criteria	Details
Purpose	Verify that unique salts are generated for password hashing.
Test steps	1. Generate multiple salts. 2. Compare the generated salts.
Expected Outcome	Generated salts should be unique and of the expected length

5.2.4 User Registration

Criteria	Details

Purpose	Verify the user registration process, including handling duplicate
Test steps	1. Register a new user. 2. Attempt to register a user with an already existing username.
Expected Outcome	1. New user registration should be successful. 2. Attempting to register a duplicate username should fail.

5.2.5 User Login

Criteria	Details
Purpose	Verify the user login process.
Test steps	1. Register a user. 2. Attempt to log in with the registered user credentials. 3. Attempt to log in with incorrect credentials.
Expected Outcome	1. Login with correct credentials should be successful. 2. Login with incorrect credentials should fai

5.2.6 Plater Authentication

Criteria	Details
Purpose	Verify the authentication of two players for a game session.
Test steps	1. Attempt to authenticate two valid players. 2. Attempt to authenticate with one or more invalid player credentials.
Expected Outcome	1. Authentication should succeed for valid player credentials. 2. Authentication should fail for invalid player credentials, with appropriate error messages.

5.3 Choose Difficulty

These tests verify the functionality related to choosing AI difficulty levels.

5.3.1 Window Initialization

Criteria	Details
Purpose	Verify the initialization of the chooseDifficulty window.
Test steps	1. Check the window geometry. 2. Verify the presence of the background image label. 3. Check that the background image is loaded correctly.
Expected Outcome	1. The window geometry is set correctly. 2. The background image label is present. 3. The background image is loaded and not null.

5.3.2 Easy Button Clicked

Criteria	Details
Purpose	Verify that clicking the Easy button correctly sets the AI difficulty.
Test steps	1. Simulate a click on the Easy button. 2. Verify that the AI difficulty is set to Easy.
Expected Outcome	The AI difficulty should be set to Easy.

5.3.3 Medium Button Clicked

Criteria	Details
Purpose	Verify that clicking the medium button correctly sets the AI difficulty.
Test steps	1. Simulate a click on the medium button. 2. Verify that the AI difficulty is set to medium.
Expected Outcome	The AI difficulty should be set to medium.

5.3.4 Hard Button Clicked

Criteria	Details
Purpose	Verify that clicking the hard button correctly sets the AI difficulty.
Test steps	1. Simulate a click on the hard button. 2. Verify that the AI difficulty is set to hard.
Expected Outcome	The AI difficulty should be set to hard.

5.3.5 Expert Button Clicked

Criteria	Details
Purpose	Verify that clicking the expert button correctly sets the AI difficulty.
Test steps	1. Simulate a click on the expert button. 2. Verify that the AI difficulty is set to expert.
Expected Outcome	The AI difficulty should be set to expert.

5.4 User Tests

These tests verify the functionality related to user profiles and statistics.

5.4.1 Initial User Stats

Criteria	Details
Purpose	Verify that a new user has correct initial statistics.
Test steps	1. Create a new user. 2. Check the user's initial total games, wins, losses, draws, AI games, player games, win rate, and best streak.
Expected Outcome	All initial statistics should be 0.

5.4.2 Password Check

Criteria	Details
Purpose	Verify that the user's password can be checked correctly.
Test steps	1. Create a user with a password. 2. Check the password with the correct password. 3. Check the password with an incorrect password. 4. Check the password with an empty string.
Expected Outcome	1. Password check should return true for the correct password. 2. Password check should return false for incorrect or empty passwords.

5.4.3 Add Win Game

Criteria	Details

Purpose	Verify that adding a won game correctly updates user statistics.
Test steps	1. Create a user. 2. Add a won game against AI. 3. Check the updated statistics (total games, wins, vs AI, win rate, best streak).
Expected Outcome	Statistics should be updated correctly to reflect the win.

5.4.4 Add Loss Game

Criteria	Details
Purpose	Verify that adding a lost game correctly updates user statistics.
Test steps	1. Create a user. 2. Add a lost game against AI. 3. Check the updated statistics (total games, losses, vs players, win rate, best streak).
Expected Outcome	Statistics should be updated correctly to reflect the loss.

5.4.5 Add Draw Game

Criteria	Details
Purpose	Verify that adding a drawn game correctly updates user statistics.
Test steps	1. Create a user. 2. Add a drawn game against AI. 3. Check the updated statistics (total games, draws, vs AI, win rate, best streak).
Expected Outcome	Statistics should be updated correctly to reflect the draw.

5.4.6 Win streak calculation

Criteria	Details
Purpose	Verify that the win streak is calculated correctly.
Test steps	1. Create a user. 2. Add a sequence of wins and losses. 3. Check the best win streak.
Expected Outcome	The best win streak should reflect the longest sequence of consecutive wins..

5.4.7 Game History

Criteria	Details
Purpose	Verify that game history is recorded and retrieved correctly.
Test steps	1. Create a user. 2. Add multiple games (win, loss). 3. Retrieve the game history. 4. Verify the size and content of the game history.
Expected Outcome	Game history should contain the correct number of games in the correct order (newest first), with accurate details for each game.

5.5 Game Mode Window

These tests evaluate the application's performance characteristics, including response times, resource utilization, and stability under stress.

5.5.1 AI Response Time

Criteria	Details
Purpose	Measure the response time of the AI opponent at different difficulty levels and game stages.
Test steps	1. Measure AI response time for Easy, Medium, Hard, and Expert difficulties. 2. Measure AI response time at early, mid, and late game stage
Expected Outcome	AI response times should be within acceptable limits (e.g., Expert AI should respond within 3 seconds).

5.5.2 Database Response Time

Criteria	Details
Purpose	Measure the response time for database operations such as saving users, loading users, and generating leaderboards.
Test steps	1. Measure time to create and save a large number of users with game data. 2. Measure time to load all users. 3. Measure time to generate the leaderboard.
Expected Outcome	Database operations should complete within reasonable timeframes (e.g., save/load within 5 seconds, leaderboard within 2 seconds).

5.5.3 Authentication Response Time

Criteria	Details
Purpose	Measure the response time for user registration, login, and password hashing.
Test steps	1. Measure baseline memory usage. 2. Measure memory usage after creating game objects. 3. Measure memory usage after creating many users with game data. 4. Measure memory usage after cleaning up objects.
Expected Outcome	Memory usage should remain within reasonable bounds, and memory should be released after objects are deleted.

5.5.4 CPU Usage

Criteria	Details
Purpose	Evaluate the CPU utilization, especially during intensive operations like AI decision-making.
Test steps	Measure CPU usage for multiple AI moves at Expert difficulty.
Expected Outcome	Average AI move time should be within acceptable limits (e.g., less than 2 seconds).

5.5.5 Game Logic Stress

Criteria	Details
Purpose	Test the game logic's stability and performance under continuous play.
Test steps	Play a large number of full games in sequence with random moves.
Expected Outcome	Games should play quickly and without crashes (e.g., average game time less than 50ms).

5.5.6 Memory Usage

Criteria	Details

Purpose	Monitor the application's memory consumption during various operations to detect potential memory leaks.
Test steps	1. Measure baseline memory usage. 2. Measure memory usage after creating game objects. 3. Measure memory usage after creating many users with game data. 4. Measure memory usage after cleaning up objects.
Expected Outcome	Memory usage should remain within reasonable bounds, and memory should be released after objects are deleted.

5.5.7 Database Stress

Criteria	Details
Purpose	Test the database's performance and stability when handling a large volume of user and game data.
Test steps	1. Create a large number of users with extensive game histories. 2. Measure time to save, load, and generate leaderboards for this large dataset.
Expected Outcome	Database operations should remain performant even with large datasets.

5.6 Integration Tests

These tests verify the seamless collaboration between different components of the Tic Tac-Toe game.

5.6.1 Authentication to Game Flow

Criteria	Details
Purpose	Verify the end-to-end flow from user authentication to starting and making moves in a game.
Test steps	1. Register and log in a new user. 2. Start a new game with the authenticated user. 3. Make a move as the authenticated user. 4. Verify that the game state updates correctly and the current player changes.
Expected Outcome	The user should be able to successfully authenticate, start a game, and make moves, with correct game state updates.

5.6.2 Game Completion to History

Criteria	Details
Purpose	Verify that game completion correctly updates user statistics and game history.
Test steps	1. Register and log in a user. 2. Record initial user statistics. 3. Play and complete a game (e.g., a win). 4. Add the game result to the user history. 5. Verify that user statistics (total games, wins/losses/draws) are updated. 6. Verify that the game is recorded in the user's game history.
Expected Outcome	User statistics and game history should be accurately updated upon game completion.

5.6.3 AI Integration

Criteria	Details
Purpose	Verify the integration between the game logic and the AI opponent.
Test steps	1. Set up a game with the AI opponent. 2. Player makes a move. 3. Verify that the AI responds with a move. 4. Verify that the correct number of moves have been made and it's the player's turn again.
Expected Outcome	The AI should correctly integrate with the game logic, making moves when it's its turn.

5.6.4 Database Integration

Criteria	Details
Purpose	Verify the integration between the application and the database for saving and loading user data and generating leaderboards.
Test steps	1. Create multiple test users with game data. 2. Save the users to the database. 3. Load users from the database. 4. Generate a leaderboard from the loaded users. 5. Verify that the data is saved, loaded, and the leaderboard is generated correctly.
Expected Outcome	User data should be persistently stored, retrieved, and used to generate accurate leaderboards.

5.7 Database Tests

These tests ensure the reliability and correctness of database operations.

5.7.1 Save and Load Users

Criteria	Details
Purpose	Verify that user data can be successfully saved to and loaded from the database.
Test steps	1. Create a set of test users with associated game data. 2. Save these users to the database. 3. Load users from the database. 4. Verify that the loaded users match the saved users.
Expected Outcome	User data should be accurately saved and retrieved, maintaining data integrity.

5.7.2 Leaderboard Generation

Criteria	Details
Purpose	Verify that the leaderboard is correctly generated based on user statistics.
Test steps	1. Provide a collection of users with varying game statistics. 2. Generate the leaderboard. 3. Verify that the leaderboard contains the correct number of entries and that each entry has accurate username, wins, total games, win rate, and best streak.
Expected Outcome	The leaderboard should be accurately generated, reflecting the users' performance.

5.7.3 Save Game

Criteria	Details
Purpose	Verify that individual game results can be saved to the database.
Test steps	1. Provide player names and game result. 2. Save the game record to the database
Expected Outcome	The game record should be successfully saved.

5.7.4 Error Handling

Criteria	Details
Purpose	Test the database's robustness and error handling capabilities when encountering invalid file paths or other issues.
Test steps	1. Attempt to save users to an invalid or unwritable file path. 2. Attempt to save users with an empty database path. 3. Attempt to save users with a database path containing invalid character
Expected Outcome	The database operations should gracefully handle errors, returning false and potentially providing error messages without crashing.

5.7.5 Large Dataset Handling

Criteria	Details
Purpose	Evaluate the database's performance and stability when handling a large number of users and extensive game histories.
Test steps	1. Create a large number of users (e.g., 100) each with a significant number of game records. 2. Save this large dataset to the database. 3. Load this large dataset from the database. 4. Generate a leaderboard using this large dataset.
Expected Outcome	The database should efficiently handle large datasets for saving, loading, and leaderboard generation without significant performance degradation or errors.

5.7.6 Leaderboard Sorting

Criteria	Details
Purpose	Verify that the leaderboard is correctly sorted based on user scores (or a similar ranking metric).
Test steps	1. Create users with different win/loss records that would result in distinct scores. 2. Generate the leaderboard. 3. Verify that the leaderboard entries are sorted in the expected order (e.g., highest score first).
Expected Outcome	The leaderboard should be correctly sorted according to the defined ranking criteria.

5.8 Authentication Tests

These tests cover the security and functionality of user authentication.

5.8.1 Password Hashing

Criteria	Details
Purpose	Verify that passwords are securely hashed using a salt.
Test steps	1. Hash the same password twice. 2. Verify that the generated hashes are different (due to salting). 3. Verify that the hash length is appropriate
Expected Outcome	Hashing should produce unique, sufficiently long hashes for the same password.

5.8.2 Password Verification

Criteria	Details
Purpose	Verify that a hashed password can be correctly verified against a plain text password.
Test steps	1. Hash a password. 2. Verify the hashed password with the correct plain-text password. 3. Verify the hashed password with an incorrect plain-text password. 4. Verify the hashed password with an empty plain-text password.
Expected Outcome	Verification should succeed for correct passwords and fail for incorrect or empty ones.

5.8.3 Salt Generation

Criteria	Details
Purpose	Verify that unique and alphanumeric salts are generated.
Test steps	1. Generate multiple salts. 2. Verify that salts are unique and have the expected length. 3. Verify that salts contain only alphanumeric characters.
Expected Outcome	Salts should be unique, of correct length, and alphanumeric.

5.8.4 User Registration

Criteria	Details
Purpose	Verify the user registration process, including handling duplicate usernames.
Test steps	1. Register a new user. 2. Attempt to register the same user again.
Expected Outcome	Initial registration should succeed; subsequent attempts with the same username should fail.

5.8.5 User Login

Criteria	Details
Purpose	Verify the user login process with correct and incorrect credentials
Test steps	1. Register a user. 2. Attempt to log in with correct credentials. 3. Attempt to log in with incorrect credentials
Expected Outcome	Login should succeed with correct credentials and fail with incorrect ones.

5.8.6 Player Authentication

Criteria	Details
Purpose	Verify the authentication of two players for a game session.
Test steps	1. Attempt to authenticate two valid players. 2. Attempt to authenticate with invalid credentials for one or both players.
Expected Outcome	Authentication should succeed for valid pairs and fail with appropriate error messages for invalid ones.

5.8.7 Empty Credentials

Criteria	Details
Purpose	Test the system's behavior when attempting to register or log in with empty usernames or passwords.
Test steps	1. Attempt registration with empty username. 2. Attempt registration with empty password. 3. Attempt login with empty username. 4. Attempt login with empty password. 5. Attempt login with both empty username and password.
Expected Outcome	All attempts with empty credentials should fail.

5.8.8 Special Characters

Criteria	Details
Purpose	Verify that the system handles special characters in usernames and passwords correctly.
Test steps	1. Register and log in with usernames containing special characters. 2. Register and log in with passwords containing special characters.
Expected Outcome	The system should successfully handle special characters in credentials.

5.8.9 Long Credentials

Criteria	Details
Purpose	Verify that the system handles very long usernames and passwords without issues
Test steps	1. Register and log in with a very long username (e.g., 100 characters). 2. Register and log in with a very long password (e.g., 100 characters).
Expected Outcome	The system should successfully handle long credentials.

5.9 AI Opponent Tests

These tests validate the behavior and performance of the AI opponent.

5.9.1 Set Game Logic

Criteria	Details
Purpose	Verify that the AI opponent can correctly be associated with a GameLogic instance and make moves within that game context.
Test steps	1. Create a new Game Logic Instance 2. Set this Game Logic Instance for the AI Opponent 3. Trigger the AI to make a move 4. Verify that move was made on the new Game Logic Board
Expected Outcome	The AI should successfully make a move on the provided GameLogic instance.

5.9.2 Make Valid Move

Criteria	Details
Purpose	Verify that the AI opponent makes a valid move on an empty or partially f illed board.
Test steps	1. Initialize a game board. 2. Trigger the AI to make a move. 3. Verify that exactly one cell has been occupied by the AI.
Expected Outcome	The AI should make a single, valid move, occupying an empty cell.

5.9.3 No Moves on Full Board

Criteria	Details
Purpose	Verify that the AI does not attempt to make a move on a full board.
Test steps	1. Fill the game board to a draw state (no empty cells, no winner). 2. Trigger the AI to make a move. 3. Verify that no changes occur on the board and the game remains in a draw state.
Expected Outcome	The AI should not make any move on a full board, and the game state should remain unchanged.

5.9.4 AI Blocks Winning Move

Criteria	Details
Purpose	Verify that the AI can correctly identify and block an opponent's winning move.
Test steps	1. Set up a game board where the player (X) has two in a row and can win on the next move. 2. Trigger the AI (O) to make a move. 3. Verify that the AI places its mark in the cell that would have completed the player's winning line.
Expected Outcome	The AI should successfully block the opponent's winning move.

5.9.5 AI Takes Winning Move

Criteria	Details
Purpose	Verify that the AI can correctly identify and take its own winning move.
Test steps	1. Set up a game board where the AI (O) has two in a row and can win on the next move. 2. Trigger the AI (O) to make a move. 3. Verify that the AI places its mark in the cell that completes its winning line and the game ends with an AI win.
Expected Outcome	The AI should successfully identify and take its winning move, leading to an AI victory.

5.9.6 AI Performance

Criteria	Details
Purpose	Evaluate the AI's performance (response time) at different difficulty levels.
Test steps	1. Measure the time taken for the AI to make a move at Easy difficulty. 2. Measure the time taken for the AI to make a move at Expert difficulty.
Expected Outcome	The AI should complete its moves within reasonable timeframes for all difficulty levels (e.g., Expert AI within 3 seconds). While Expert might take longer than Easy, both should be performant.