1.Test Setup

initTestCase: Runs once before all tests begin. It creates a single instance of the TicTacToe game object and establishes the connection to the in-memory test database.

init: Runs before *each* test case. It calls resetGame() to provide a clean board and clears all database tables to ensure tests are independent and do not influence each other.

cleanupTestCase: This function runs once after all tests are complete. It safely deletes the game object and removes the test database connection to free up resources.

2.Detailed Test Case Analysis

A) Game Basics

1.testInitialState

Purpose: To confirm that a new game always starts in a clean, predictable state.
Algorithm:
calls resetGame().
The test loops through all 9 board positions.
It asserts that each position contains an empty character (' ').

"Pass" Indicates: resetGame() function is successfully clearing the board.

2,3.testWinCondition & testWinCondition_Diagonal

Purpose: To verify that the checkWin() logic correctly identifies different winning patterns.

Algorithm:

It simulates a sequence of moves by calling the private makeMove() function to create a guaranteed win for the starting player (both horizontally and diagonally).

It asserts that the checkWin() function returns true for that player.

"Pass" Indicates: The win-detection is working correctly for multiple scenarios.

4.testDrawCondition

same as 2 and 3 but tests the checktie() instead of checkwin()

5.testInvalidMoveIsIgnored

Purpose: confirm game prevents players from making a move on an already occupied square.

Test Algorithm:

Makes a valid move in the center of the board. try to make a second move in same place

Asserts that the board state and the current player have not changed.

"Pass" Indicates: The makeMove() function correctly blocks invalid moves.

B) AI Logic Tests

6.testAiMakesBlockingMove

Purpose: verify that the "Hard" AI plays defensively to block an opponent's imminent win.

Test Algorithm: Uses a helper function (setTestBoardState) to create a specific board where the opponent ('O') has two in a row. Calls makeAIMove().

Asserts that the AI placed its 'X' in the one correct square to block the opponent from winning.

"Pass" Indicates: The hardMove() and minimax() functions correctly identified the threat and made the optimal defensive move.

7.testAiMakesWinningMove

Purpose: verify that the "Hard" AI plays offensively to win when given the opportunity.

Test Algorithm: similar to last test but this time puts AI in a possible win situation

Asserts that the AI placed its 'X' in the third square to complete the line and win.

"Pass" Indicates: The hardMove() and minimax() functions correctly identified an opportunity to win and took it.

C) Test PVP mode

8.testPlayerVsPlayerFlow

Purpose: confirm that in a Player-vs-Player match, the win counter is correctly incremented for the winning player.

Test Algorithm: Sets the game mode to PvP. Plays out a winning sequence of moves.

Asserts that the correct player's win counter has increased by exactly 1.

A "Pass" Indicates: The score-keeping logic is working correctly for a standard PvP game.

D) Test Series of games to win option

9.testSeriesEndLogic

Purpose: verify the "best of" series logic

Test Algorithm: Sets up a "best of 3" game where a player has 1 win. It calls resetGame() and asserts the board was cleared for the next game.

It then sets the score to 2 wins, puts marks on the board, and calls resetGame() again.

It asserts that the board was **not** cleared, proving the game correctly identified the end of the series.

"Pass" Indicates: The game correctly handles the termination logic for a series of matches.

E) Login and Authentication Tests and Database

10.testGuestLogin

Purpose: To verify that the "Play as Guest" feature correctly sets the application's internal state.

Test Algorithm: Calls the guestLogin() function.

Asserts that the guestMode flag is true and the loggedInUser is set to the default guest name.

A "Pass" Indicates: The guest login flow is working as expected.

11,12,13.testSuccessfulRegistration & testFailedRegistration_WhenUserExists & testUserRegistrationInDatabase()

Purpose: Ensure new users can be created and system correctly prevents duplicate usernames.

Test Algorithm: The success test simulates typing a new username/password, calls registerAccount(), and then queries the database to assert that the user now exists.

The failure test calls registerAccount() twice with the same information and asserts that the user count in the database is still only 1.

"Pass" Indicates: The registerAccount() function is correctly interacting with the database and enforcing unique username constraints.

14,15.testSuccessfulLogin & testFailedLogin_WithWrongPassword

Purpose: To verify the complete login workflow for registered users.

Test Algorithm: The success test registers a user, then simulates typing correct credentials and calls handleLogin(). It asserts that the internal loggedInUser state is correct.

The failure test does the same but provides the wrong password, asserting that the loggedInUser state does **not** change.

"Pass" Indicates: The login logic is correctly validating credentials against the database.

16.testGameSaveToDatabase

Purpose: confirm that a completed game's result is correctly saved to the database.

Test Algorithm: Sets up player names and a move history. Calls the saveIndividualGame() function with a "Win" result.

Directly queries the database to assert that the saved row contains the correct winner, result, and move history.

"Pass" Indicates: The game's results and statistics are being correctly saved for later viewing.

F) Test the Game themes

17.testThemeApplication

Purpose: Ensure that the applyStyleSheet() function, which contains a large amount of complex CSS code, can run without errors and successfully applies a style to the window.

Test Algorithm:
Manually sets the game->selectedTheme variable to "Dark".
Calls the game->applyStyleSheet() function.
Retrieves the styleSheet() property from the main game window.
Asserts that the retrieved stylesheet string is not empty.

"Pass" Indicates: theme-switching logic is functional

G) Integration Tests

18.testIntegration_ButtonClick

Purpose: verify that the UI and game logic are correctly integrated by checking if clicking a game button updates both the visual display and the internal game state.

Algorithm:

Finds a specific game grid QPushButton by its unique name.
Verifies the button was found and its text is initially empty.
Simulates a mouse click on the button.
Asserts that the button's visible text has updated to the correct player's symbol (the UI check).
Asserts that the internal board state in the logic has also updated at the corresponding position (the logic check).

"Pass" Indicates: The UI buttons are correctly connected to the game logic, and a single user action correctly updates all necessary parts of the system.

19.testIntegration_GameSettings

Purpose: Test the integration of settings screen UI with game's internal configuration variables.

Algorithm:
- Navigates to the settings screen, Finds the QSpinBox widgets for "Total Games" and "Games to Win" and the "Apply" button by their unique names.
- Programmatically sets new values in the spin boxes.
- Simulates a click on the "Apply Settings" button.
- Asserts that the internal game state variables (totalGames, gamesToWin) have been updated to match the values set in the UI.

"Pass" Indicates: UI controls on the settings screen are correctly connected to and successfully modify the application's core settings.

20.testIntegration_RegistrationAndLoginFlow

Purpose: Test the end-to-end integration of the entire user authentication system, from creating an account to successfully logging in.

Algorithm:
- Simulates typing a new username and password and clicking "Register".

- Asserts the user was successfully created in the database.

- Simulates typing the same credentials again and clicking "Login".

- Asserts that the internal application state now reflects a successful login and the screen has changed

"Pass" Indicates: The entire user-facing authentication workflow, which involves multiple UI elements, function calls, and database interactions, works together seamlessly.

21. testMatchHistoryPopulation

Purpose: To test the integration between the database saving feature and the UI that displays the match history.

Algorithm:
- Programmatically saves several game results to the test database.
- simulates a click on the "View Match History" button.
- Asserts that the application navigates to the history screen.
- Asserts that the number of rows in the table exactly matches the number of games that were saved.

"Pass" Indicates: The loadMatchHistory() function is correctly querying the database and populating the UI table with the results, showing that the saving and loading features are well-integrated.

Performance Test

22.testPerformance_HardAiMove

Purpose: To benchmark the "response time" of the most computationally expensive function in the game: the "Hard" difficulty AI's decision-making process.

Test Algorithm:
-Test sets up a typical mid-game board scenario for a Player-vs-AI match on "Hard" difficulty.
-It creates a Timer and starts it immediately before calling the makeAIMove() function.

"Pass" Indicates: The AI was able to successfully calculate and make its move within the acceptable time limit.

Also The result of this test is the benchmark number printed to the output