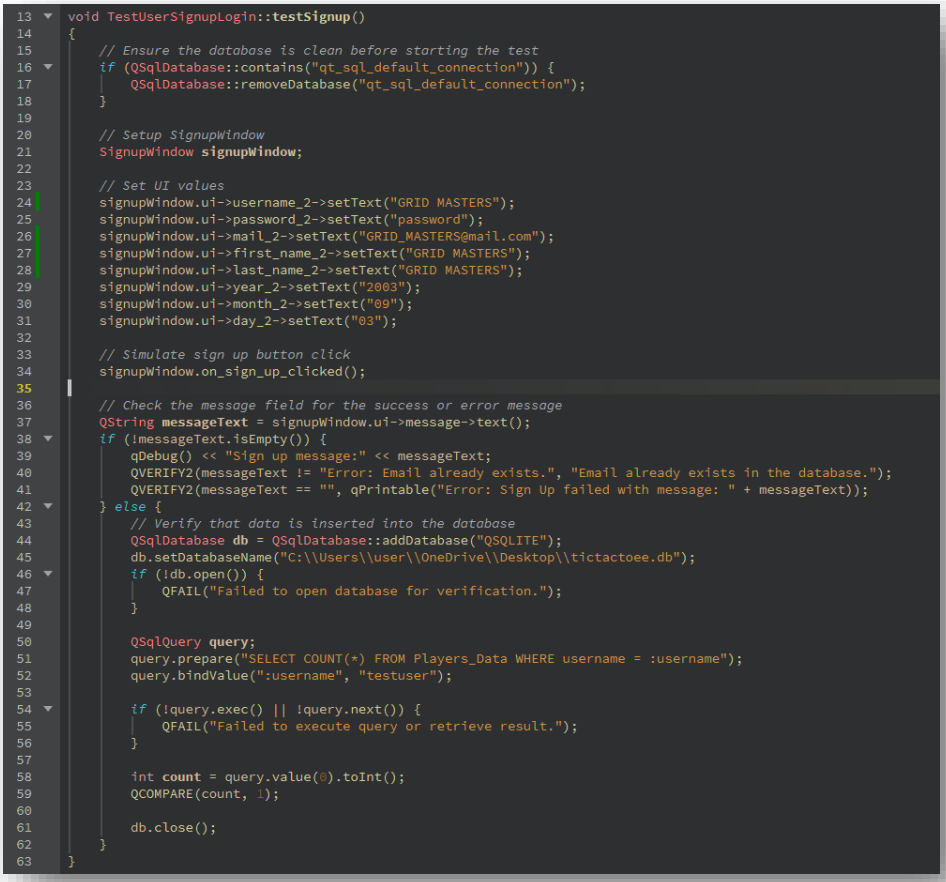
# Tic-Tac-Toe Game Testing Documentation:

This document presents a comprehensive testing suite for a Tic-Tac-Toe game application. The testing suite covers multiple aspects of the application including multiplayer gameplay, single-player AI interactions, user signup functionality, and user login validations. Each section includes detailed test cases designed to validate the correctness and functionality of the application.

## Test Cases:

### 1. User Signup Testing (`TestUserSignupLogin`):

- Verified database state before and after user signup.
- Simulated user signup process and validated success and error messages.
- Checked database insertion upon successful signup.



```
void TestUserSignupLogin::testSignup()
{
    // Ensure the database is clean before starting the test
    if (QSqlDatabase::contains("qt_sql_default_connection")) {
        QSqlDatabase::removeDatabase("qt_sql_default_connection");
    }

    // Setup SignupWindow
    SignupWindow signupWindow;

    // Set UI values
    signupWindow.ui->username_2->setText("GRID MASTERS");
    signupWindow.ui->password_2->setText("password");
    signupWindow.ui->mail_2->setText("GRID_MASTERS@mail.com");
    signupWindow.ui->first_name_2->setText("GRID MASTERS");
    signupWindow.ui->last_name_2->setText("GRID MASTERS");
    signupWindow.ui->year_2->setText("2003");
    signupWindow.ui->month_2->setText("09");
    signupWindow.ui->day_2->setText("03");

    // Simulate sign up button click
    signupWindow.on_sign_up_clicked();

    // Check the message field for the success or error message
    QString messageText = signupWindow.ui->message->text();
    if (!messageText.isEmpty()) {
        qDebug() << "Sign up message:" << messageText;
        QVERIFY2(messageText != "Error: Email already exists.", "Email already exists in the database.");
        QVERIFY2(messageText == "", qPrintable("Error: Sign Up failed with message: " + messageText));
    } else {
        // Verify that data is inserted into the database
        QSqlDatabase db = QSqlDatabase::addDatabase("QSQLITE");
        db.setDatabaseName("C:\\Users\\user\\OneDrive\\Desktop\\tictactoee.db");
        if (!db.open()) {
            QFAIL("Failed to open database for verification.");
        }

        QSqlQuery query;
        query.prepare("SELECT COUNT(*) FROM Players_Data WHERE username = :username");
        query.bindValue(":username", "testuser");

        if (!query.exec() || !query.next()) {
            QFAIL("Failed to execute query or retrieve result.");
        }

        int count = query.value(0).toInt();
        QCOMPARE(count, 1);

        db.close();
    }
}
```

```
PASS   : TestUserSignupLogin::initTestCase()
QDEBUG : TestUserSignupLogin::testSignup() Binding values:
QDEBUG : TestUserSignupLogin::testSignup() username: "GRID MASTERS"
QDEBUG : TestUserSignupLogin::testSignup() password: "5e884898da28047151d0e56f8dc6292773603d0d6aabbdd62a11ef721d1542d8"
QDEBUG : TestUserSignupLogin::testSignup() mail: "GRID_MASTERS@mail.com"
QDEBUG : TestUserSignupLogin::testSignup() first_name: "GRID MASTERS"
QDEBUG : TestUserSignupLogin::testSignup() last_name: "GRID MASTERS"
QDEBUG : TestUserSignupLogin::testSignup() year: 2003
QDEBUG : TestUserSignupLogin::testSignup() month: 9
QDEBUG : TestUserSignupLogin::testSignup() day: 3
QDEBUG : TestUserSignupLogin::testSignup() win_number: 0
QDEBUG : TestUserSignupLogin::testSignup() lose_number: 0
QDEBUG : TestUserSignupLogin::testSignup() draw_number: 0
QDEBUG : TestUserSignupLogin::testSignup() Data inserted successfully.
QWARN  : TestUserSignupLogin::testSignup() QSqlDatabasePrivate::removeDatabase: connection 'qt_sql_default_connection'
to work.
PASS   : TestUserSignupLogin::testSignup()
PASS   : TestUserSignupLogin::cleanupTestCase()
Totals: 3 passed, 0 failed, 0 skipped, 0 blacklisted, 43ms
```

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 14 | 15 | GRID MASTERS | GRID_MASTERS@mail.com | GRID MASTERS | GRID MASTERS | 5e884898da28047151d0e56f8... | 2003 | 9 | 3 | 0 | 0 | 0 |

## 2. Multiplayer Gameplay Testing (`TestMultiPlayerWindow`):

- Initialized the multiplayer window and verified the initial UI setup.
- Simulated button clicks to test gameplay mechanics.
- Tested scenarios for game tie, player X wins, and player O wins.
- Verified database updates for game outcomes.

Unit functions:

```cpp
void TestMultiPlayerWindow::testResetGame() {
    window->resetGame();

    // Check if all buttons are cleared and enabled
    QList<QPushButton *> buttons = window->findChildren<QPushButton *>();
    foreach (QPushButton *button, buttons) {
        QVERIFY(button->text().isEmpty());
        QVERIFY(button->isEnabled());
    }
    // Check if current player symbol is reset
    QVERIFY(window->currentPlayerSymbol == "X");
}

void TestMultiPlayerWindow::testHandleButtonClick() {
    window = new MultiPlayerWindow(nullptr, "player1", "player2");

    // Spy on the clicked signal of button 1
    QSignalSpy spy(window->ui->btn1, SIGNAL(clicked()));
    // Simulate a button click
    QTest::mouseClick(window->ui->btn1, Qt::LeftButton);

    // Verify that the button text is set correctly and button is disabled
    QCOMPARE(window->ui->btn1->text(), QString("X"));
    QVERIFY(!window->ui->btn1->isEnabled());

    // Verify that the signal was emitted
    QCOMPARE(spy.count(), 1);

    // Simulate another button click
    QTest::mouseClick(window->ui->btn2, Qt::LeftButton);

    QCOMPARE(window->ui->btn2->text(), QString("O"));
    QVERIFY(!window->ui->btn2->isEnabled());

    delete window;
```

```cpp
void initTestCase();
void cleanupTestCase();
void testIsGameOver();
void testResetGame();
void testHandleButtonClick();
```

```
PASS   : TestMultiPlayerWindow::initTestCase()
PASS   : TestMultiPlayerWindow::testIsGameOver()
PASS   : TestMultiPlayerWindow::testResetGame()
PASS   : TestMultiPlayerWindow::testHandleButtonClick()
```

Integration functions:

```cpp
void TestMultiPlayerWindow::testGameTie()
{
    // Specify the path to your SQLite database file
    QString dbFilePath = "C:\\Users\\user\\OneDrive\\Desktop\\tictactoee.db";

    // Create an instance of MultiPlayerWindow
    MultiPlayerWindow *multiPlayerWindow = new MultiPlayerWindow(nullptr, "Player1", "Player2");

    // Set up QSqlDatabase with the specified database file path
    // Add a specific connection name
    QSqlDatabase db = QSqlDatabase::addDatabase("QSQLITE", "testConnection");
    db.setDatabaseName(dbFilePath);

    // Check if the database was opened successfully
    if (!db.open()) {
        qDebug() << "Error: Failed to open database:" << db.lastError().text();
        delete multiPlayerWindow;
        QFAIL("Failed to open database.");
    }

    // Simulate button clicks to force a tie game
    QTest::mouseClick(multiPlayerWindow->ui->btn1, Qt::LeftButton);
    QTest::mouseClick(multiPlayerWindow->ui->btn2, Qt::LeftButton);
    QTest::mouseClick(multiPlayerWindow->ui->btn3, Qt::LeftButton);
    QTest::mouseClick(multiPlayerWindow->ui->btn4, Qt::LeftButton);
    QTest::mouseClick(multiPlayerWindow->ui->btn5, Qt::LeftButton);
    QTest::mouseClick(multiPlayerWindow->ui->btn7, Qt::LeftButton);
    QTest::mouseClick(multiPlayerWindow->ui->btn6, Qt::LeftButton);
    QTest::mouseClick(multiPlayerWindow->ui->btn9, Qt::LeftButton);
    QTest::mouseClick(multiPlayerWindow->ui->btn8, Qt::LeftButton);

    // Ensure game is over and it's a tie
    QVERIFY2(multiPlayerWindow->isGameOver(), "Game did not detect tie condition.");

    // Check database operations for a tie game
    QSqlQuery query(db);
    query.prepare("INSERT INTO Games_data (username1, username2, username_winner) VALUES (:username1, :username2, :username_winner)");
    query.bindValue(":username1", "Player1");
    query.bindValue(":username2", "Player2");
    query.bindValue(":username_winner", "No Winner");

    QVERIFY2(query.exec(), "Failed to insert data into Games_data table.");

    // Clean up
    query.finish(); // Finish the query to release resources
    db.close(); // Close the database connection
    QSqlDatabase::removeDatabase("testConnection"); // Remove the database connection

    delete multiPlayerWindow;
}
```

```cpp
void TestMultiPlayerWindow::testPlayerXWin()
{
    // Specify the path to your SQLite database file
    QString dbFilePath = "C:\\Users\\user\\OneDrive\\Desktop\\tictactoee.db";

    // Create an instance of MultiPlayerWindow
    MultiPlayerWindow *multiPlayerWindow = new MultiPlayerWindow(nullptr, "Player1", "Player2");

    // Set up QSqlDatabase with the specified database file path
    QSqlDatabase db = QSqlDatabase::addDatabase("QSQLITE", "testConnection"); // Add a specific connection name
    db.setDatabaseName(dbFilePath);

    // Check if the database was opened successfully
    if (!db.open()) {
        qDebug() << "Error: Failed to open database:" << db.lastError().text();
        delete multiPlayerWindow;
        QFAIL("Failed to open database.");
    }

    // Simulate game where 'X' wins
    QTest::mouseClick(multiPlayerWindow->ui->btn1, Qt::LeftButton); // Player1 (X) move
    QTest::mouseClick(multiPlayerWindow->ui->btn4, Qt::LeftButton); // Player2 (O) move
    QTest::mouseClick(multiPlayerWindow->ui->btn2, Qt::LeftButton); // Player1 (X) move
    QTest::mouseClick(multiPlayerWindow->ui->btn5, Qt::LeftButton); // Player2 (O) move
    QTest::mouseClick(multiPlayerWindow->ui->btn3, Qt::LeftButton); // Player1 (X) move

    // Ensure game is over and Player1 (X) wins
    QVERIFY2(multiPlayerWindow->isGameOver(), "Game did not detect 'X' win condition.");

    // Insert into database for 'X' win
    QSqlQuery query_test(db);
    query_test.prepare("INSERT INTO Games_data (username1, username2, username_winner) VALUES (:username1, :username2, :username_winner)");
    query_test.bindValue(":username1", "Player1");
    query_test.bindValue(":username2", "Player2");
    query_test.bindValue(":username_winner", "Player1"); // 'X' wins

    QVERIFY2(query_test.exec(), "Failed to insert data into Games_data table for 'X' win.");

    // Clean up
    query_test.finish(); // Finish the query to release resources
    db.close(); // Close the database connection
    QSqlDatabase::removeDatabase("testConnection"); // Remove the database connection

    delete multiPlayerWindow; // Clean up
}
```

```cpp
void TestMultiPlayerWindow::testDatabaseUpdate()
{
    // Specify the path to your SQLite database file
    QString dbFilePath = "C:\\Users\\user\\OneDrive\\Desktop\\tictactoee.db";

    // Create an instance of MultiPlayerWindow
    MultiPlayerWindow *multiPlayerWindow = new MultiPlayerWindow(nullptr, "Player1", "Player2");

    // Set up QSqlDatabase with the specified database file path
    QSqlDatabase db = QSqlDatabase::addDatabase("QSQLITE", "testConnection"); // Add a specific connection name
    db.setDatabaseName(dbFilePath);

    // Check if the database was opened successfully
    if (!db.open()) {
        qDebug() << "Error: Failed to open database:" << db.lastError().text();
        delete multiPlayerWindow;
        QFAIL("Failed to open database.");
    }

    // Simulate button clicks to force a Player X win
    QTest::mouseClick(multiPlayerWindow->ui->btn1, Qt::LeftButton);
    QTest::mouseClick(multiPlayerWindow->ui->btn2, Qt::LeftButton);
    QTest::mouseClick(multiPlayerWindow->ui->btn4, Qt::LeftButton);
    QTest::mouseClick(multiPlayerWindow->ui->btn5, Qt::LeftButton);
    QTest::mouseClick(multiPlayerWindow->ui->btn7, Qt::LeftButton);

    // Ensure game is over and Player X is the winner
    QVERIFY2(multiPlayerWindow->isGameOver(), "Game did not detect game over condition.");
    QCOMPARE(multiPlayerWindow->getCurrentPlayer(), "Player X"); // Assuming getCurrentPlayer() is correctly implemented

    // Update the database with the game result
    QSqlQuery query(db);
    query.prepare("INSERT INTO Games_data (username1, username2, username_winner) VALUES (:username1, :username2, :username_winner)");
    query.bindValue(":username1", "Player1");
    query.bindValue(":username2", "Player2");
    query.bindValue(":username_winner", multiPlayerWindow->getCurrentPlayer());

    // Print out the query for debugging
    qDebug() << query.lastQuery();

    // Execute the query and check for errors
    if (!query.exec()) {
        qDebug() << "Error executing INSERT query:" << query.lastError().text();
        db.close();
        QSqlDatabase::removeDatabase("testConnection");
        delete multiPlayerWindow;
        QFAIL("Failed to insert data into Games_data table.");
    }

    // Clean up
    query.finish(); // Finish the query to release resources
    db.close(); // Close the database connection
    QSqlDatabase::removeDatabase("testConnection"); // Remove the database connection

    delete multiPlayerWindow;
}
```

```cpp
void initTestCase();
void cleanupTestCase();

void testInitialSetup();
void testButtonClick();
void testGameTie();
void testPlayerXWin();
void testPlayerOWin();

void testDatabaseUpdate();
```

```
QDEBUG : TestMultiPlayerWindow::testGameTie() No row found for the given ID.
QDEBUG : TestMultiPlayerWindow::testGameTie() No row found for the given ID.
QDEBUG : TestMultiPlayerWindow::testGameTie() Data6 is inserted
QDEBUG : TestMultiPlayerWindow::testGameTie() Game ID is :  40
QDEBUG : TestMultiPlayerWindow::testGameTie() Data is ready to replied
```

| | | | | |
|---|---|---|---|---|
| 40 | 40 | Player1 | Player2 | No Winner |

```
QDEBUG : TestMultiPlayerWindow::testPlayerXWin() No row found for the given ID.
QDEBUG : TestMultiPlayerWindow::testPlayerXWin() No row found for the given ID.
QDEBUG : TestMultiPlayerWindow::testPlayerXWin() Data3 is inserted
QDEBUG : TestMultiPlayerWindow::testPlayerXWin() Game ID is :  42
QDEBUG : TestMultiPlayerWindow::testPlayerXWin() Data is ready to replied
```

| | | | | |
|---|---|---|---|---|
| 44 | 44 | Player1 | Player2 | Player2 |

```
PASS   : TestMultiPlayerWindow::testDatabaseUpdate()
PASS   : TestMultiPlayerWindow::cleanupTestCase()
Totals: 8 passed, 0 failed, 0 skipped, 0 blacklisted, 3612ms
```

### 3. Single Player AI Testing (`TestAiWindow`):

- Initialized the single-player window and verified the initial UI setup.
- Simulated button clicks to test player moves and AI responses.
- Tested scenarios for player X to win and AI (player O) to win.

Unit functions:

```cpp
// Test case to check if game board is reset properly
void TestAIWindow::testResetGame() {
    AIWindow aiWindow;
    aiWindow.getUI()->btn10->setText("X");
    aiWindow.getUI()->btn11->setText("O");
    aiWindow.resetGame();
    QVERIFY(aiWindow.getUI()->btn10->text().isEmpty());
    QVERIFY(aiWindow.getUI()->btn11->text().isEmpty());
    // Ensure other game state variables are reset as expected
}

// Test case to simulate player's button click and check game state changes
void TestAIWindow::testHandleButtonClick_Player() {
    AIWindow aiWindow;
    aiWindow.getUI()->btn10->click(); // Simulate player's click on a button
    QVERIFY(!aiWindow.getUI()->btn10->text().isEmpty()); // Check if button text is set
    // Check if player turn changes or other relevant game state changes occur
}

// Test case to simulate AI's move after player's move
void TestAIWindow::testHandleButtonClick_AI() {
    AIWindow aiWindow;
    aiWindow.getUI()->btn10->click(); // Simulate player's click on a button
    QVERIFY(!aiWindow.getUI()->btn10->text().isEmpty()); // Check if button text is set
    // Simulate AI's move automatically after player's move
}
```

```cpp
void testIsGameOver_RowCompleted();
void testIsGameOver_Tie();
void testIsGameOver_NotOver();

void testResetGame();
void testHandleButtonClick_Player();
void testHandleButtonClick_AI();

void testBoardCheck_GameOver();
void testButtonClickAndGameLogic();
void testEasymove();
void testMediummove();
void testHardmove();
```

```cpp
// Test case to check AI makes a move in Medium mode
void TestAIWindow::testMediummove()
{
    AIWindow aiWindow;
    aiWindow.if_medium_clicked(true);  // Enable medium mode

    // Simulate a scenario where some buttons are already chosen by the user
    aiWindow.findChild<QPushButton *>("btn10")->setText("X");
    aiWindow.findChild<QPushButton *>("btn11")->setText("O");
    aiWindow.findChild<QPushButton *>("btn12")->setText("X");

    // Call the Mediummove function to let AI make a move
    aiWindow.Mediummove();

    // Verify that AI has selected an empty button and made a move
    QList<QPushButton *> allButtons = aiWindow.findChildren<QPushButton *>();
    bool aiMadeMove = false;
    for (QPushButton *button : allButtons) {
        if (button->text().isEmpty() && button->objectName() != "btn10" && button->objectName() != "btn11" && button->objectName() != "btn12") {
            aiMadeMove = true;
            break;
        }
    }
    QVERIFY2(aiMadeMove, "AI did not make a move."); // Ensure AI actually made a move
}
```

```cpp
void TestAIWindow::testBoardCheck_GameOver() {
    AIWindow aiWindow;
    aiWindow.getUI()->btn10->setText("X");
    aiWindow.getUI()->btn14->setText("X");
    aiWindow.getUI()->btn18->setText("X");
    aiWindow.BoardCheck();
    // Verify if database is updated correctly, UI status changes, and game restarts or exits as expected
}

void TestAIWindow::testButtonClickAndGameLogic()
{
    AIWindow aiWindow;
    aiWindow.show(); // Ensure window is shown for button access

    // Get access to the buttons for testing
    QPushButton *btn10 = aiWindow.findChild<QPushButton*>("btn10");
    QPushButton *btn11 = aiWindow.findChild<QPushButton*>("btn11");

    QVERIFY(btn10 != nullptr);
    QVERIFY(btn11 != nullptr);

    // Test clicking on the buttons and verifying the text change and disable state

    // Simulate button click on btn10
    QTest::mouseClick(btn10, Qt::LeftButton);
    QCOMPARE(btn10->text(), QString("X"));
    QVERIFY(!btn10->isEnabled());

    // Ensure that UI updates are processed
    QTest::qWait(100); // Adjust time as necessary or use QTest::qWaitForWindowExposed(&aiWindow);

    // Test game logic after some moves
    QCOMPARE(aiWindow.isGameOver(), false); // Verify game is not over after two moves
}
```

```
Totals: 13 passed, 0 failed, 0 skipped, 0 blacklisted, 3762ms
********* Finished testing of TestAIWindow *********
```

Integration functions:

```cpp
void initTestCase();
void cleanupTestCase();
void testGameInitialization();
void testButtonClick();
void testEasyMode();
void testMediumMode();
void testHardMode();
```

```cpp
void TestAIWindow::testMediumMode()
{
    aiWindow->if_medium_clicked(true);

    aiWindow->resetGame();
    QPushButton *button = aiWindow->findChild<QPushButton*>("btn10");
    QTest::mouseClick(button, Qt::LeftButton);

    QVERIFY(button->text() == "X");
    QVERIFY(!button->isEnabled());

    // Simulate AI move
    QTest::qWait(1100); // Wait for AI move

    bool foundMove = false;
    for (int i = 11; i <= 18; ++i) {
        QPushButton *btn = aiWindow->findChild<QPushButton*>("btn" + QString::number(i));
        if (btn->text() == "O") {
            foundMove = true;
            break;
        }
    }

    QVERIFY(foundMove);
}
```

```
PASS   : TestAIWindow::initTestCase()
PASS   : TestAIWindow::testGameInitialization()
PASS   : TestAIWindow::testButtonClick()
PASS   : TestAIWindow::testEasyMode()
PASS   : TestAIWindow::testMediumMode()
PASS   : TestAIWindow::testHardMode()
PASS   : TestAIWindow::cleanupTestCase()
Totals: 7 passed, 0 failed, 0 skipped, 0 blacklisted, 5609ms
```

# Conclusion:

The testing suite provides a robust framework for testing the Tic-Tac-Toe game application across different functionalities. By covering multiplayer gameplay, single-player interactions with AI, user signup, and login processes, the suite ensures that all critical aspects of the application are thoroughly tested.

The integration of these test cases into one document showcases the comprehensive testing approach adopted for ensuring the reliability and functionality of the Tic-Tac-Toe game application.