Cairo University

Faculty of Engineering

Electronics and Electrical Communications Engineering

TIC TAC TOE

TESTING DOCUMENTATION

Supervised by: Dr. **Omar Nasr**

• Introduction:

➢ **Google test**: Google Test (gtest) is a C++ testing framework developed by Google. It's widely used for writing unit tests, integration tests "Ideal for core logic and non-GUI code".

➢ **Qt test**: Qt Test is the official testing framework provided by Qt. For components that are heavily based on Qt types and involve GUI elements, Qt Test is the preferred framework, as it provides seamless support for Qt-specific features such as signals, slots, widgets, and event handling.

There are two main types of tests commonly used to ensure code quality and system reliability:

- **Unit testing:**

  Unit testing involves testing individual components or functions of a program in isolation. The goal is to ensure that each unit of the software performs as expected.

- **Integration testing:**

  Integration testing focuses on verifying that multiple components of the system work together correctly. It checks interactions between modules, such as UI, logic, and data.

1. Unit testing:

➢ **Testing the Tic-Tac-Toe Game Logic:** "test_tictactoegame.cpp" :
covers core game logic (move validation – win/tie detection - AI decisions)

- Test Framework: Google Test

- Test Functions:

➢ makeMove(int row, int col, Player player):

• Functionality in code: Attempts to place a player's mark on a given cell. Returns true if successful, false if the cell is occupied or out of bounds.

- Test Cases:

Test Name	Description	Result
MakeMove_ValidCell_ReturnsTrue	Test Valid move updates board.	● Passed
MakeMove_AlreadyFilled	Move rejected on already filled cell	● Passed
MakeMove_OutOfBoundsNegative	Move rejected at negative index	● Passed
MakeMove_OutOfBounds	Move rejected outside board size (≥ 3)	● Passed

➢ checkWin(Player player):

• Functionality in code: Determines if the given player has a winning configuration (3 in a row, column, or diagonal).

• Test Cases:

Test Name	Description	Result
CheckWin_RowWin_ReturnsTrue	Detects row win condition	🟢 Passed
CheckWin_ColumnWin_ReturnsTrue	Detects Column win condition	🟢 Passed
CheckWin_MainDiagonal_ReturnsTrue	Detects main diagonal win condition	🟢 Passed
CheckWin_AntiDiagonal_ReturnsTrue	Detects anti-diagonal win condition	🟢 Passed
CheckWin_NoWin_ReturnsFalse	No win when board is not aligned	🟢 Passed

➢ Minimax(…):

• Functionality in code: Recursive decision algorithm for optimal AI moves, with alpha-beta pruning.

• Test Cases:

Test Name	Description	Result
AIWinScore	Detects when AI wins; the returned score = 10 "Depth = 0 in code"	🟢 Passed
HumanWinScore	Detects when Human wins; the returned score = -8 "Depth = 2 in code"	🟢 Passed
DrawScore	Detects draw on full board: score = 0	🟢 Passed

➢ switchPlayer():

• Functionality in code: This function toggles the value of the currentPlayer between HUMAN and AI.

• Test Cases:

– Initial Player is HUMAN:

Verifies that when the game starts, the current player is HUMAN.

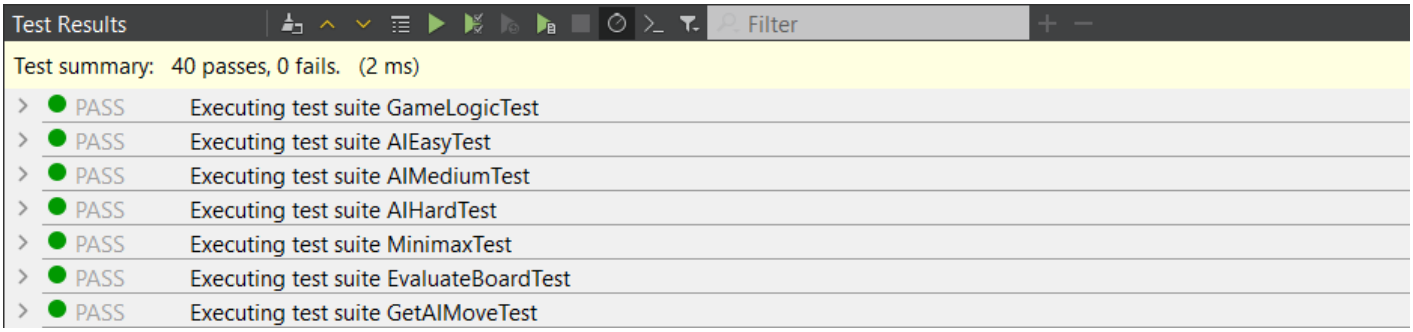
– First Call to switchPlayer() Changes Player to AI:

Ensures that after one call to switchPlayer(), the current player becomes AI

– Second Call to switchPlayer() Changes Player Back to HUMAN:

Confirms that another call toggles the player back to HUMAN.

➢ Test Results:



➢ **Testing the TestMainWindow Class** :"test_mainwindow.cpp" :

mainwindow.cpp is the central controller for your game's user interface.
It manages screen transitions, user input, gameplay visuals, animations, and integrates with backend logic from UserManager and TicTacToeGame.

• Test Framework: Qt test

• Test Functions:

➢ MainWindow(QWidget *parent):

• Functionality in code: It ensures that the window is successfully created (not a null pointer), the title is set to the expected login prompt (" 🎮 Tic Tac Toe - Sign In to Play!"), and that the window remains hidden by default until explicitly shown. This confirms correct UI initialization before user interaction begins.

• Test Function name : testMainWindowInitialization()

• Test Cases:
– The test confirms that the MainWindow object is successfully instantiated.
– It verifies that the window title is correctly set to " 🎮 Tic Tac Toe - Sign In to Play!"
– It checks that the window is not visible immediately after creation

➢ showLoginScreen():
• Functionality in code:
switches the main interface to display the login page by setting the stacked widget to loginPage. It updates the window title to prompt the user to sign in and clears the username and password input fields to ensure a clean state each time the login screen is shown.

• Test Function Name: testLoginScreenNavigation()

• Test Cases:

– Verifies the window title is correctly set when the login screen is shown.

– Checks that the register navigation button labeled **“New Player”** or **“Join”** is present on the login screen.

– Verifies that the window title changes to " 📝 Tic Tac Toe - Create Account!" when the register button is clicked.

➢ setupSymbolWidget():

• Functionality in code: This function is called during setupUI() and creates the symbol selection radio buttons. It defines the UI behavior that lets the user pick whether they play as X or O.

• Test Function Name: testSymbolSelection()

• Test Cases:

– Finds the radio buttons used to choose between symbol **X** and **O**.

– Simulates clicking **X**, then verifies only X is selected.

– Simulates clicking **O**, then verifies only O is selected.

– Confirms mutual exclusivity of the two options.

➢ **Testing the UserManager Class** “test_usermanager.cpp”:

The UserManager class provides user account management, authentication, game history, and data persistence for your Tic Tac Toe application. It uses SQLite as a backend and handles:

– 🔐 User Registration & Authentication

– 🔑 Password Security

– 📦 User Data Management

– 🕹️ Game History Storage

– 💾 Database Initialization

– 🧹 Utility Methods: Lists all usernames - check if a user exists - Manages session/logout state

• Test Framework: Qt test

• Test Functions:

➢ hashPassword():

• Functionality in code: takes a plain-text password and returns a secure, hex-encoded SHA-256 hash. This ensures that passwords are never stored or compared in plain text in the database or memory.
It is a core security utility used during: user registration and user login

• Test Function Name: testPasswordHashingConsistency()

- Test Cases:

– Hashed passwords are stored and used consistently: Both users are able to log in using the same plain-text password, confirming that hashPassword() creates a consistent hash that can be compared on login.

– No cross-user interference:
Logging in with consistency1, then consistency2, verifies that hashing is consistent and user-specific storage works independently.

➢ userExists():

- Functionality in code:
checks if a given username is already present in the database.

- Test Function Name: testUserExists()

- Test Cases:

– Verifies if function returns true for an existing user.

– Verifies if function returns false for a non-existing user.

➢ getAllUsernames:

- Functionality in code: retrieves a list of all usernames stored in the users table, sorted alphabetically.

- Test Function Name: testGetAllUsernames()

- Test Cases:

– The test creates 3 users (user1, user2, user3) and verifies that these usernames appear in the returned list. This confirms that getAllUsernames() successfully reads the database after inserts.

– It checks that the number of usernames increases after new users are added.
This confirms that the function reflects real-time data and isn't returning a cached or outdated list.

➢ **Testing the TestUser Class** "test_user.cpp":

User class acts as the in-memory model for a player in your Tic Tac Toe application. It holds all relevant user data and methods related to user stats, history, and identity.
Core Functional areas:

– 🎮 Game Statistics: it tracks the number of (wins/losses/ties) then returns the percentage of games won out of total games played.

– 📋 Game History.

– 📄 Identity & Profile Info

– 🛠️ Constructors : Default constructor initializes a blank user and Parameterized constructor allows setting username, password, email, and sets lastLogin to currentDateTime().

- Test Framework: Qt test

- Test Functions:

➢ getWinRate()

- Functionality in code:
  It returns the percentage of games the user has won, based on all games played.
- Test Function Name: testGetWinRateZeroGames()
- Test Cases:

– Verifies that when a user has played 0 games, getWinRate() should return 0.0

➢ addGameToHistory(const GameRecord &record)

- Functionality in code:
  This function adds a game result to the user's game history, by inserting the given GameRecord at the beginning of the gameHistory list.
- Test Function Name: testAddGameToHistory()
- Test Cases:

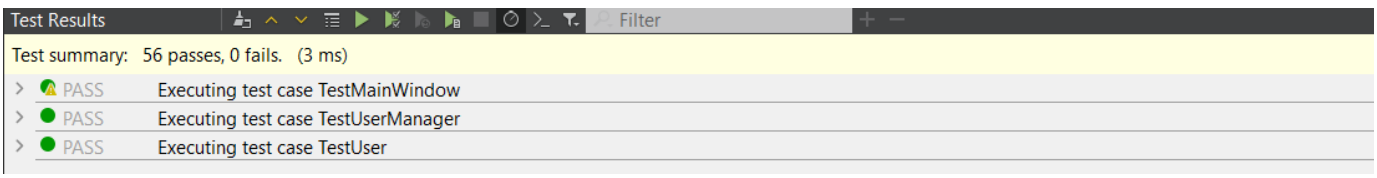
– Verify that adding a game record increases the game history size by 1
This ensures that addGameToHistory() properly appends the new record to the user's in-memory history list.

➢ User()

- Functionality in code:
  It is a default constructor for the User class. It is automatically called when you create a User object without passing any parameters
- Test Function Name: testDefaultConstructor()
- Test Cases:

– getGamesWon() returns 0 for a new user.
– getGamesLost() returns 0 for a new user.
– getGamesTied() returns 0 for a new user.
– getUsername() is empty for a new user.
This test confirms that the default constructor creates a clean user object.

➢ Test Results for all Qt tests files:



2. Integration Testing:

Test Name	Validates	Covered Files
RegisterLoginLogoutEndToEnd()	User registration, login, logout.	usermanager, user, mainwindow.
PvPGame_RecordsHistoryForLoggedUser()	PvP game logic, move recording, win detection, result saved in history	mainwindow, tictactoegame, user.
PvAI_Easy_CompletesWithoutCrash()	PvAI game flow, board logic, easy AI, game ends (win/lose/tie) without crashing	mainwindow, tictactoegame.

➢ Test Results: