GUI QA Instructions & Manual Testing Guide

This document is intended for the GUI team and future QA testers. It outlines the manual testing procedures required whenever a change is made to any part of the GUI. This is particularly useful for ensuring visual and functional consistency across the application.

GUI Test Case 1: Login Window – Successful Login
Objective: Verify that a user can log in with valid credentials and is routed to the Main Menu.
Steps:

1. Launch the application (initiated by HelloApplication).
2. In the Login Window, enter a valid, registered username (found inside temp.txt)
3. Enter the correct corresponding password
4. Click the "Login" button.

Expected Result:

- A success message is displayed.
- The Main Menu window (MainMenuWindow) loads

Pass/Fail: Pass
Notes:

- Ensure that user authentication is validated (see LoginWindowController.java for the login flow).
- Verify that the stage transition is smooth and that UI elements (e.g., images and animations) load correctly.

GUI Test Case 2: Login Window – Empty Fields
Objective: Ensure that the system prevents login when required fields are left blank.
Steps:

1. Launch the Login Window (HelloApplication)
2. Leave the username field blank and/or the password field blank.
3. Click the "Login" button.

Expected Result:

- An error alert appears stating that username and/or password cannot be empty.
- No transition to another screen occurs.

Pass/Fail: Pass

Notes:

- Check that the error message matches the text defined in LoginWindowController.java.

GUI Test Case 3: Login Window – Incorrect Password / Nonexistent Username
Objective: Validate that the application shows appropriate error messages when incorrect data is provided.
Steps:

1. Launch the Login Window.
2. Enter a non-existent username or an incorrect password for an existing username.
3. Click "Login."

Expected Result:

- An error message is displayed indicating either the username was not found or the password is incorrect.
- No further navigation occurs.
  Pass/Fail:
  Notes:
- Verify that the error content is clear and that the UI remains responsive.

GUI Test Case 4: Login Window – Guest Login
Objective: Confirm that guest login functions correctly and bypasses standard authentication.
Steps:

1. Launch the Login Window.
2. Click the "Join as Guest" (or equivalent) button.

Expected Result:

- The application logs the user in as a guest.
- The Main Menu is launched with a default guest profile.

Pass/Fail: Pass

GUI Test Case 5: Sign-Up Window – Field Validation (Invalid Email / Weak Password)
Objective: Ensure that the sign-up form rejects invalid input, such as an improperly formatted email or a short password.
Steps:

1. Launch the Sign-Up Window from the Login Window.
2. In the username, email, and password fields, provide an invalid email (e.g., "userATexample") and/or a password shorter than the minimum length.
3. Click the "Register" button.

Expected Result:

- An error alert pops up indicating the specific validation issue (e.g., "Invalid Email" or "Weak Password").
- No account is created, and the sign-up window remains active for corrections.

Pass/Fail: Pass

Notes:

- Check that validations in SignUpWindowController.java (including email format and password length) work as specified.
- Edge case: Ensure error is triggered even if only one of the validations fails.

GUI Test Case 6: Sign-Up Window – Successful Account Creation
Objective: Verify that a new user can create an account with valid information.
Steps:

1. Launch the Sign-Up Window.
2. Enter a unique username, a valid email address, and a strong password (minimum eight characters).

3. Click “Register.”

Expected Result:

- A success alert is displayed confirming account creation.
- The application transitions to the Main Menu with the newly created user profile loaded.

Pass/Fail: Pass
Notes:

- Confirm that the transition uses OpenMainMenu() in SignUpWindowController.java.
- Check for proper error handling if the username already exists.
- If the user selects return to login, the GUI should transition back to the Login window

GUI Test Case 7: Forgot Password Window – Missing Fields
Objective: Confirm that all required fields are validated when resetting the password.
Steps:

1. From the Login Window, click the “Forgot Password” link.
2. In the Forgot Password Window, leave one or more fields (username, email, new password, or confirm password) empty.
3. Click the “Reset Password” button.

Expected Result:

- An error alert is displayed stating that all fields must be filled in.

Pass/Fail: Pass

Notes:

- Match the error messages with those defined in ForgetPasswordWindowController.java.

GUI Test Case 8: Forgot Password Window – Password Mismatch and Weak Password
Objective: Validate that the system checks for both password strength and consistency.
Steps (Password Mismatch):

1. From the Login Window, click the "Forgot Password" link.
2. Enter valid username and email.
3. Enter a new password and a different confirmed password.
4. Click "Submit."

Expected Result (Mismatch):

- An error alert indicating "Passwords do not match" is displayed.

Steps (Weak Password):

1. Enter valid username, email, and matching passwords that are less than eight characters long.
2. Click "Submit."

Expected Result (Weak Password):

- An error alert indicates that the password must be at least eight characters long.

Pass/Fail: Pass
Notes:

- Ensure both validations trigger the correct error messages**.**

GUI Test Case 9: Forgot Password Window – Successful Password Reset
Objective: Verify that the password reset functionality works when provided with correct information.
Steps:

1. Launch the Forgot Password Window.
2. Fill in all fields with valid data and matching, sufficiently strong passwords.
3. Click "Submit."

Expected Result:

- An information alert confirms that the password was changed successfully.
- The window transitions (or instructs the user) to log in using the new password.

Pass/Fail: Pass

Notes:

- Check the transition logic in ForgetPasswordWindowController.java for scene change.

GUI Test Case 10: Main Menu – Navigation and Game Selection
Objective: Verify that the Main Menu displays all game selection options and navigates correctly.
Steps:

1. Log in and load the Main Menu.
2. Verify that game cards for Tic-Tac-Toe, Connect Four, and Checkers are visible.
3. Click on each game card and observe the action (e.g., clicking "View Rules" on a card or the card itself to start a game).
4. Use the "Find Match" option to test matchmaking functionality.

Expected Result:

- The Main Menu loads without UI glitches.
- Each game card responds correctly (launching the respective game window or showing rules).
- The matchmaking dialog appears, and, if not cancelled, the appropriate game window is launched (see MainMenuWindow.java).

Pass/Fail: Fails
Notes:

- **Current iteration does not allow for users to return to main menu after clicking on "View Rules"**

GUI Test Case 11: Leaderboard – Sorting, Searching, and Reset Functionality
Objective: Ensure that the Leaderboard window displays player stats accurately and that sorting, searching, and reset features work.
Steps:

1. Navigate to the Leaderboard screen from the Main Menu via clicking “View Full Leaderboard”
2. Verify that the table is populated with player rankings.
3. Use the sort choice box to change sort criteria (e.g., sort by “Total wins” or other metrics).
4. Enter a search query in the search field and click “Search.”
5. Click “Reset” to clear the search filter.

Expected Result:

- The table updates dynamically based on the sort option.
- Only the player matching the search query is shown after searching.
- The “Reset” operation restores the full leaderboard.

Pass/Fail: Pass

Notes:

- Verify that podium positions (first, second, third) are updated correctly (see LeaderBoardController.java).

GUI Test Case 12: Game Window – Game Board and Move Validation (Tic-Tac-Toe Example)
Objective: Confirm that the game window renders correctly and that moves are processed accurately.
Steps:

1. From the Main Menu, launch a multiplayer Tic-Tac-Toe game.
2. Observe that the game board (a 3x3 grid) is set up with blank cells.
3. Click on an empty cell to make a move.
4. Verify that the cell is updated with the correct symbol (“X” or “O”) based on the turn and that the turn indicator changes appropriately (top right).

5. Try clicking on an already filled cell and check that no move is registered.

6. Complete moves to simulate a win, or draw state.

Expected Result:

- The game board is rendered without layout issues.
- Moves are correctly registered and displayed.
- The game detects wins/draws and shows an appropriate dialog with game over messages.

Pass/Fail: Pass
Notes:

- Similar test cases should be devised for Connect Four and Checkers (see GameWindow.java for board setup logic).
- Include checks for the computer move in "player vs. computer" mode

GUI Test Case 13: Game Window – Chat Functionality
Objective: Verify that the in-game chat functions properly, including message entry, auto-update, and animation (simulated "Typing…" effects).
Steps:

1. Launch a game that supports chat (All games support chat).

2. Type a message in the chat input and click "Send."

3. Observe the message appending to the chat display area and the simulated "Typing…" animation for the computer's response.

Expected Result:

- User messages appear immediately, and after a delay, the computer's response replaces the "Typing…" indicator with the bots message
- No UI elements overlap or break during the animation.

Pass/Fail: Pass

Notes:

- Verify the timeline animation and update mechanism as implemented in the GameWindow's chat module.

GUI Test Case 14: User Profile Window – Tab Navigation and Data Display
Objective: Ensure that the User Profile window displays various tabs (Overview, Game Stats, Match History, Settings) correctly and that data visualizations (e.g., pie charts) populate as expected.
Steps:

1. From the Main Menu, navigate to the User Profile window (via the "Profile" button).

2. Click through each tab (Overview, Game Stats, Match History, Settings).

3. Verify that each tab shows correctly formatted content:
   – Overview: Welcome message, stats grid, and game distribution pie chart.
   – Game Stats: Correct win/loss/draw charts and a table with game records.
   – Match History: table with recent match details.
   – Settings: Change password, Notification settings options, and bug report

Expected Result:

- All tabs are accessible without UI glitches.

- Charts and tables display mock/demo data appropriately.

- Changes made in the Settings tab (if applicable) are reflected on save.

Pass/Fail: Pass
Notes:

- Test responsiveness when resizing the window and verify consistency with the dark theme.

GUI Test Case 15: Scene Transitions and Consistent Styling
Objective: Confirm that scene transitions (managed by SceneManager) occur smoothly and that the visual style (colors, fonts, spacing) is consistent across all windows.

Steps:

1. Navigate through the application from login to various screens (Main Menu, Game Window, Leaderboard, User Profile, etc.).

2. Observe the transition animation or delay between scene changes.

3. Verify that the stylesheet (e.g., userprofile.css referenced in SceneManager.java) is consistently applied.

Expected Result:

- All scene transitions are smooth with no flickering or loading delays.

- The visual style (background colors, fonts, and spacing) is uniform across windows.

Pass/Fail: Pass
Notes:

- Test on multiple screen resolutions by changing the size of the windows

- Verify that critical elements (e.g., buttons, labels) have proper padding and alignment.

GUI Test Case 16: Rules Window – Tab Display and Return Button

Objective: Ensure that the rules windows (Checkers, Connect Four, and Tic-Tac-Toe) open correctly, tabs are responsive, and the "Back" button properly returns the user to the Main Menu.

Steps:

1. From the Main Menu, click on the “More Info” button for any of the games (e.g., Checkers, Connect Four, Tic-Tac-Toe).

2. Confirm that the “More Info” page opens.

3. Resize the window and check that the tabs dynamically adjust width (minimum 100px each).

4. Click through each tab and confirm that content is readable and styled correctly.

5. Press the "Back" button.

Expected Result:

- Rules window launches with the correct FXML layout.
- All tabs are visible and evenly distributed.
- Tab width adjusts dynamically with window resizing.
- Clicking "Back" closes the rules window and returns the user to the Main Menu.

Pass/Fail:
Pass

Notes:

- Tab resizing logic is handled by `updateTabWidths()` in each rules controller class.
- Each controller (`TicTacToeRules`, `CheckersRules`, `ConnectFourRules`) uses a similar layout structure, so you can reuse this test case for all three.
- Ensure the `Stage` is properly injected with `setStage()` in each controller from the scene loader.