GUI Manual Testing

CHECKERS GUI

Test: Test valid move highlighting
Objective: Verify that when a checker piece is selected, all valid destination tiles are highlighted on the board to indicate legal moves.

Setup/Steps:

1. Launch the application and using MainApplication and navigate to start a checkers game
2. Ensure the game is in an active state and it is one player's turn (e.g., P1).
3. Click on a piece that has valid moves (e.g., a piece with a diagonal empty square or possible capture).
4. After selecting the piece, visually confirm that the valid destination tiles are highlighted including possible pieces that can be played, the destinations those pieces can move, etc.
5. If a piece must perform a capture (according to rules), ensure that only capturing moves are highlighted.
6. Select another piece that has no legal moves and verify that no highlight appears.
7. Click outside the board or on an invalid square and confirm that all highlights disappear.

Expected Result:
Upon selecting a checker piece with valid moves, only the valid destination squares are highlighted. If a capture is available, only capturing destinations are highlighted and selecting invalid pieces or clicking elsewhere resets the highlights.

Pass/Fail:
Test has passed if only legal move destinations are highlighted when a movable piece is clicked and highlight disappears when clicking outside or after moving. Test fails if incorrect tiles are highlighted, no tiles are highlighted when they should be, or tiles remain highlighted incorrectly after an invalid interaction

Notes:
If the test fails, check the refresh board method in CheckersGUIController and the getBoardCells in the CheckersController for game logic.

Test: Test turn indicator updates

Objective: Verify that the turn indicator display updates to show the correct player's turn after a piece is played (i.e. after player blue places a piece, a pink piece is displayed on the screen).

Setup/Steps:

1. Launch the application and navigate to a game of checkers.
2. Verify that the screen is visible and is displaying a players turn
3. Get initial value of the getCurrentPlayer in checkers game logic verify that the correct banner image is being loaded onto the screen.
4. Simulate a player move by moving a valid piece diagonally.
5. After the simulated move, get updated value of gameLogic.getCurrentPlayer()
6. Ensure that the screen was updated to reflect the current player after the piece was played.

Expected Result: The screen switches to the other player's coloured piece and the correct banner is displayed on the screen (i.e. switches from pink to blue after pink has played a turn).

Pass/Fail: Test has passed if the GUI displays the correct player's turn (via its corresponding banner) to both player's screens, false otherwise

Notes:
If test fails, check:

- refreshBoard and getTurn in CheckersGUIController

TIC-TAC-TOE GUI

Test: Test turn indicator

Objective: Test the X's turn and O's turn banner is switching properly after every piece played.

Steps:

1. Launch application and navigate to tic tac toe
2. X's turn should be initialized
3. Get current player, should be X
4. Place a piece on the board (verify it is an X piece)
5. Check that turn indicator now says O's turn
6. After game is over check turn indicator is null

Expected Result: Turn indicator switches for every piece placed and should disappear after a game ends.

Pass/Fail: Passes if X's turn is initialized, turn indicator switches after every piece becoming the new current player, and disappears after game is over.

Test: Chat messages

Objective: Test that messages from the local chat field are shown on chat, and opponent chat messages are shown.

Steps:

1. Launch tic tac toe
2. Set the players chat field to a message
3. Use send message function
4. Verify that chat window has appended proper message locally
5. Use a received message function with a message from opponent
6. Verify that chat window has appended opponents message

Expected Result: Player can view their own messages and opponents messages in chat

Pass/Fail: Passes if messages in the chat window match test messages for both player and opponent.

Test: Info display

Objective: Test that info button is working

Steps:

1. Launch tic tac toe
2. Run the info button click event
3. Check that the info display imageview is showing the info screen
4. Run the info display close mouse click event
5. Verify that info display imageview is null

Expected Result: Info display should appear and disappear when the info button is clicked, and when ok button is clicked respectively.

Pass/Fail: Passes if info imageview is set to info display after first mouse click event, then null after second mouse click event.

Test: Mute button

Objective: Test that audio manager is reacting to mute button events

Steps:

1. Launch tic tac toe
2. Run mute button click event
3. Verify that audio manager mute status = muted
4. Run the mute button click event again
5. Verify that audio manager mute status = unmuted

Expected Result: Audio manager mutes and unmutes after mute button clicks, soundtracks should also be passing due to audio manager changed

Pass/Fail: Passes if audio manager status = muted after first mute button click event, and unmuted after second mute button click event.

Test: Update GUI

Objective: Test that GUI is being updated properly from a board parameter

Steps:

1. Launch tic tac toe
2. Run updateBoard with a known board as parameter
3. Verify that each piece on the tic tac toe board is properly set

Expected Result: When updateBoard runs, the board on the GUI side is properly updated to reflect the input board.

Pass/Fail: Passes if every piece on tic tac toe board after updateBoard matches input board.

CONNECT 4 GUI

1. **Test**: Test turn indicator updates

Objective: Verify that the turn indicator display updates to show the correct player's turn after a piece is played (i.e. after player blue places a piece, "pink's turn" banner is displayed at the top of the interface).

Setup/Steps:

Launch the application and initialize the C4GUIController
Verify that the turnIndicator is visible (and not null)
Get initial value of Client2.getC4CurrentPlayer() and verify that the correct banner image is being loaded into the turnIndicatorImage imageview

- If current player is C4Piece.RED (i.e. last played piece was by player blue), pink_turn.png should be loaded into turnIndicatorImage
- If current player is C4Piece.BLUE (i.e. last played piece was by player pink), blue_turn.png should be loaded into turnIndicatorImage

Simulate a player column clock by invoking the onCol(col)Click() method (0 <= col <= 6)
After the simulated click, get updated value of Client2.getC4CurrentPlayer()
Ensure that turnIndicatorImage was updated to reflect the current player after the piece was played (i.e. the value loaded into turnIndicatorImage should have switched to the opposite image compared to its initial value)

Expected Result: The updateTurnIndicator() method switches to the other player's color and the correct banner is displayed on the screen (i.e. switches from pink to blue after pink has played a turn).

Pass/Fail: Test has passed if the GUI displays the correct player's turn (via its corresponding banner) to both player's screens, false otherwise

Notes:

- If test fails, check:
  - updateTurnIndicator() method in C4GUIController class and all instances where it is called within the class
  - Check getCurrentPlayer() method (turn indicator feature may not be working if current player's turn is not being correctly received by the controller class)

2. **Test**: Test hint display (win hint)

Objective: Verify that the hint column is highlighted in yellow and that C4_hint_win_image.png is displayed on the screen when a "WIN" HintResult is returned by game logic code.

Setup/Steps:

1) Launch the application and initialize the C4GUIController
2) Simulate a game state where a win is possible during the current player's turn (i.e. player currently has three discs in a row without having been blocked by their opponent, they can place a fourth disc to get four in a row, and it is currently their turn)
3) Check the value of the returned HintResult object from Client2.getC4ColHint()
    - Check that col != -1 (if this is the case, game logic could not find a valid win or block hint)
    - Check that type is "WIN"
4) Confirm that the correct column is highlighted (column that player must choose to get four in a row)
5) Check that the hintMessageImage is visible and that its value is set to the correct source (C4_hint_win_image.png).

Expected Result: The correct column is highlighted yellow, and the win hint image is displayed to the user. They can click "OK" on the banner to have this message disappear, and the yellow column highlight will be removed.

Pass/Fail: Test has passed if the GUI displays the win hint banner if the current player selects the hint button while they have three discs in a row and can place a fourth one to win the game

Notes:

- If the highlight does not appear, check highlightHintColumn()
- If the image does not appear, check clickHintButton() logic
- This part requires game logic code to function; if the wrong image appears, review getC4HintColumn() in C4GameLogic.

3. **Test**: Interface updates after each player move

Objective: Verify that each time a player makes a move, the GUI updates the c4GUIGrid to display the appropriate piece (pink or blue) in the correct cell.

Setup/Steps:

1) Begin game of Connect 4
2) Record the current state of the board array (using Client2.getC4Board())
3) Click a column on the GUI

4) The handleColumnClick method is called, which invokes Client2.receiveInput() to process the move
5) The board is updated by updateBoard() using the new game state
6) Check that the grid now contains a new piece in the correct cell based on the players move (either a pink piece or a blue piece)

Expected Result: A piece image (either pink or blue) has been added to the correct cell in the correct column (based on the column between 0 and 6 that the most recent player has chosen for their move)

Pass/Fail: Test has passed if the GUI displays the correct colored piece in the correct cell (in the correct column) after each player's turn. Test fails if wrong colored piece is placed, piece is placed in wrong cell/column, or piece is not placed in grid at all after player move.

Notes:

- This test tests integration between game logic and visual updates based on the board state
- Test all seven columns and moves within each cell within each of these columns for a more comprehensive test
- If test fails, check connections between Client2.receiveInput(), C4Piece[][], and updateBoard() in C4GUIController

4. **Test**: Test chat message display

Objective: Check that when a player enters a valid chat message, the message appears in the chat area display and the input field is cleared and reset

Setup/Steps:

1) Run game of Connect 4 so that chatField, chatArea, and sendButton for Connect 4 are initialized
2) Type a valid message (i.e. one that is not inappropriate) into the chatField input box
3) Click the sendButton
4) sendMessage() (invoked by clicking sendButton) makes sure that the message is not empty, and the message is appropriate (i.e. ChatManager.isAppropriate(message) returns true)
5) If the message is valid, the text is appended to the chatArea
6) The chatField input box is cleared

Expected Result: The chat message appears in chatArea, appended to "You: ," and ending with a newline so that the next message sent appears on the next line. The chatField input box is cleared after the send button is clicked.

Pass/Fail: Test has passed if the GUI displays the message entered into the input box by the user (so long as it is appropriate) appended to "You: ," in the chat area. The test fails if the valid message entered by the user does not appear in the chat box.

Notes:

- This test ensures that the text input is validated and displayed in the chat

5. **Test**: Multiplayer network (in the case that the network was implemented)

   Objective: Test that the GUI correctly syncs player moves, turn indicator banners, and chat functionality between two connected clients during a multiplayer game.

   Setup/Steps:

   1) Launch the application on two separate client accounts (both connected to the same session/server)
   2) Make a move on the first client's account
   3) On the second client's account, check the board grid to ensure that the piece played by the first client appears in the same cell
      - Check that the c4GUIGrid display updates to match the display of the first client after their move
   4) For both clients, run additional tests to verify that the turnIndicator image updates to correctly reflect the updated value of the current player's turn (check that it updates on both client's interfaces)
   5) Test that the send chat function words by sending a message from the first client to the second client.
      - Check that the message appears in the first client's chat area as a sent message
      - Check that the message appears in the second client's chat area as a received message
   6) Test that the reverse of the previous steps also work for communications/plays that begin with the second client
   7) Continue playing game until game is tied or won
   8) Test that the resultImage appears on both client accounts and that the column buttons are disabled on both sides once a game has ended

Expected Result: Moves made by one player appear accurately on the other player's screen (both players see the same game state on the board grid at any given time throughout the game). Turn indicator updates correctly for both players. Chat messages are synced so that they appear in the correct format on each client's interface. Endgame result is displayed accordingly on both user's screens.

Pass/Fail: Test has passed if the expected results for each individual player appear on each client's screen. Test has failed if game updates/results are not visually updated accurately on either client's screen.

Notes:

- Repeat tests all column moves for both clients to ensure greater coverage
- Network message handling may be responsible for potential failed tests

LOGIN PAGE

Test: Test successful user login

Objective:

Verify that when valid credentials are entered into the login form, the user is successfully authenticated and directed to the main application interface.

Setup/Steps:

1) Launch the application
2) Navigate to the login screen
3) Enter a valid username and password
4) Click the "Login" button
5) Wait for the response from the server or local authentication logic
6) Check if the user is redirected to the main screen/dashboard
7) User can choose to play a game or view leaderboard

Expected Result:

User is logged in successfully, the main interface is displayed, and no error messages appear.

Pass/Fail:

Test has passed if the user is directed to the main application screen and sees a personalized welcome message. Fails if the user remains on login screen or sees an error message despite valid credentials.

Notes:
If the test fails, check:

- Authentication logic
- Connection to user database
- Proper handling of redirection post-login

Test: Test unsuccessful user login

Objective:

Verify that when invalid credentials are entered, the application shows an appropriate error message and prevents access to the main interface.

Setup/Steps:

1) Launch the application
2) Navigate to the login screen
3) Enter an invalid username or incorrect password
4) Click the “Login” button
5) Wait for authentication response
6) Check for presence of an error message (e.g., “Incorrect username or password”)
7) Ensure the main interface is not loaded

Expected Result:

The application shows an error message indicating invalid credentials and does not allow access to the main interface.

Pass/Fail:

Test has passed if an error message appears and the user remains on the login screen. Fails if access is granted or error is not displayed.

Notes:

If test fails, check:

- Error handling in the login logic

- Proper verification in user authentication methods
- Input validation for username and password fields

Test: Test account creation flow

Objective:

Verify that a new user can successfully create an account, receive confirmation, and proceed to the main interface or login screen.

Setup/Steps:

1) Launch the application
2) Navigate to the “Sign Up” screen
3) Enter valid information in all required fields
4) Click the “Create Account” button
5) Confirm user is either redirected to the login screen or logged in automatically

Expected Result:

User account is created and the user proceeds to login or main screen depending on the app flow.

Notes:

If test fails, check:

- Account creation method
- Validation logic for field inputs
- Username uniqueness checks
- Password matching checks

Leaderboard

Test: Test sort by individual games

Objective:
Verify when selecting to sort by a specified game, it outputs the correct game statistics

Setup/Steps:

1)Launch the application

2) Navigate to the leaderboard

3)Click on specified game button (connect 4, tic tac toe, or checkers)

- Sorted by rating by default (expected)

4)Compare results with output from Leaderboard.java and LeaderboardTest.java

5)Repeat for the other games

Expected Result:
Leaderboard table should output the statistics (rating, wins, win/loss ratio) from the correct game selected

Pass/Fail:
Test has passed if the user is shown leaderboard output, sorted by the correct game selected. Fails otherwise.

Notes:
If the test fails, check:

- Leaderboard sorting logic
- Leaderboard processing
- GUI controller logic
- .fxml file fxids

Test: Test sort by specified statistics

Objective:
Verify when selecting to sort by a specified game, it outputs the correct player statistics

Setup/Steps:

1)Launch the application

2) Navigate to the leaderboard

3)Click on specified game button (connect 4, tic tac toe, or checkers)

4)Select sort by rating, wins, or win loss ratio

4)Compare results with output from Leaderboard.java and LeaderboardTest.java

5)Repeat for the other statistics sorting options

Expected Result:
Leaderboard table should output the statistics from the correct sorting option selected

Pass/Fail:
Test has passed if the user is shown leaderboard output, sorted by the correct option selected. Fails otherwise.

Notes:
If the test fails, check:

- Leaderboard sorting logic
- Leaderboard processing
- GUI controller logic
- .fxml file fxids

Test: Test toggle sort order

Objective:
Verify toggling sort order reverses the order of the leaderboard.

Setup/Steps:

1)Launch the application

2) Navigate to the leaderboard

3)Click on specified game button (connect 4, tic tac toe, or checkers)

4)Review descending order statistics

5)Press “toggle sort order”

6)Review ascending order statistics

7)Repeat steps 4-6 as necessary

Expected Result:
Leaderboard table should output the statistics in the reverse order originally shown

Pass/Fail:
Test has passed if the user is shown leaderboard output, sorted in the opposite order than originally shown. Fails otherwise.

Notes:
If the test fails, check:

- Leaderboard sorting logic
- Leaderboard processing
- GUI controller logic
- .fxml file fxids

Test: Test filter friends

Objective:
Verify that the leaderboard correctly outputs the friends list upon selecting “filter friends”.

Setup/Steps:

1)Launch the application

2) Navigate to the leaderboard

3)Click on specified game button (connect 4, tic tac toe, or checkers)

4)Filter friends via drop down menu

5)Compare output with friend's list

Expected Result:
Leaderboard table should output only the player's friends

Pass/Fail:
Test has passed if the user is shown only their friends in the leaderboard menu. Fails otherwise.

Notes:
If the test fails, check:

- Leaderboard sorting logic
- Leaderboard processing
- GUI controller logic
- .fxml file fxids

Test: Test search player

Objective:
Verify that the leaderboard correctly outputs the searched player upon request.

Setup/Steps:

1)Launch the application

2) Navigate to the leaderboard

3)Click on specified game button (connect 4, tic tac toe, or checkers)

4)Type in a player username in the search box, then press “search”

5)Compare output with searched player username

Expected Result:
Leaderboard table should output only the search player’s username

Pass/Fail:
Test has passed if the user is shown only the searched player's username, or if the username doesnt exist, then output “player not found”. Fails otherwise.

Notes:
If the test fails, check:

- Leaderboard sorting logic
- Leaderboard processing
- GUI controller logic
- .fxml file fxids