Project Milestone #2

Task #1: Create a testing plan – Define a testing plan or, in other words, a series of test cases that characterize the current issue. Collectively, these test cases should:

* Identify the current issue (i.e., at least one proposed test should fail)
* b) Completely describe the desired behavior of the system (i.e., all proposed tests should pass once the current issue is resolved).

Describe each test case using the “given, when, then” template introduced in class and in Assignment 2. You do NOT need to implement your test cases in an automated testing framework (e.g., JUnit).

**Issue #1: Reversed Turn Order After Host Attempts Early Move - Corrective**

**a. Identify current issues:**

**Test Case 1: Host initiates the game without an opponent.**

Given: The game is open, and the host attempts to start the game without an opponent.

When: The host clicks the "Start Game" button.

Then: Verify that the "NO\_OPPONENT" error message is displayed.

**Test Case 2: Host attempts an early move before an opponent joins.**

Given: The game is open, and the host has started the game without an opponent.

When: The host tries to make a move before an opponent joins.

Then: Confirm that the system displays the "NOT\_YOUR\_TURN" error message for the host.

**Test Case 3: Opponent joins after the host's early move attempt.**

Given: The game is in a state where the host has attempted an early move without an opponent, and the opponent joins.

When: Another player joins the game.

Then: Observe that the "NOT\_YOUR\_TURN" error appears for the host, indicating the disrupted turn order.

**b. Desired Behavior After Issue Resolution:**

**Test Case 4: Correct host turn order after issue resolution.**

Given: The game is open, and the host has attempted an early move without an opponent.

When: After the issue is resolved.

Then: Confirm that the host can make the first move, and the turn order is correct, following the expected gameplay sequence.

**Test Case 5: Opponent makes the first move after issue resolution.**

Given: The game is in a state where the host has attempted an early move without an opponent, and the opponent joins.

When: After the issue is resolved.

Then: Confirm that the host can make the first move, and the turn order is correct, with the host having the first turn as intended.

**Test Case 6: Verify no interference with turn order in a standard game.**

Given: A standard game with both host and opponent present.

When: Players take turns making moves.

Then: Confirm that the turn order remains consistent, and there is no interference or disruption.

**Issue #2: Unrecognized Victory: Missing Winner Announcement After Line Completion - Corrective**

**a. Identify current issues:**

**Test Case 1: Unrecognized Victory**

Given the game is in progress, the host completes a winning line.

When the host completes a winning line (diagonal).

Then verify that the UI displays "No Winner for this Game" despite the completion of a winning line, and the game state shows "gameOver: true."

**Test Case 2: Unrecognized Victory - Other Winning Scenarios**

Given the game is in progress, the host and opponent are actively playing.

When the host completes a winning line in other scenarios (row or column).

Then verify that the UI incorrectly displays "No Winner for this Game," and the game state shows "gameOver: true."

**Test Case 3: Unrecognized Victory - Game State Validation**

Given the game is in progress, the host and opponent are actively playing.

When the game state shows "gameOver: true" after the host completes a winning line.

Then verify that the UI incorrectly displays "No Winner for this Game."

**b. Desired Behavior After Issue Resolution:**

**Test Case 4: Recognized Victory Announcement**

Given the game is in progress, the host and opponent are actively playing.

When the host completes a winning line (diagonal).

Then after the issue is resolved, confirm that the UI correctly announces the winner, and the game state shows "gameOver: true."

**Test Case 5: Recognized Victory Announcement - Other Winning Scenarios**

Given the game is in progress, the host and opponent are actively playing.

When, after the issue is resolved, the host completes a winning line in other scenarios (row or column).

Then confirm that the UI correctly announces the winner, and the game state shows "gameOver: true."

**Test Case 6: Accurate Game State and Victory Announcement**

Given the game is in progress, the host and opponent are actively playing.

When after the issue is resolved, the game state shows "gameOver: true" after the host completes a winning line.

Then confirm that the UI accurately announces the winner, and the game state shows "gameOver: true."

**Issue 3: Misleading "PLACEMENT\_CONFLICT" and "NOT\_YOUR\_TURN" Errors Triggered by Specific Move Sequence - Corrective**

**a. Identify current issues:**

**Test Case 1: Initial PLACEMENT\_CONFLICT Error**

Given: The game is in progress, with the host and opponent actively playing.

When: The host and opponent follow the specified sequence of moves, leading to the opponent attempting to place "O" at (1,1).

Then: Verify that the game triggers an initial "PLACEMENT\_CONFLICT" error when attempting to place "O" at (1,1).

**Test Case 2: Subsequent PLACEMENT\_CONFLICT Errors**

Given: The game is in progress, with the host and opponent actively playing.

When: The host and opponent encounter the initial "PLACEMENT\_CONFLICT" error at (1,1).

Then: Verify that subsequent attempts to place marks at (1,1) result in additional "PLACEMENT\_CONFLICT" errors.

**Test Case 3: Misleading "NOT\_YOUR\_TURN" Error**

Given: The game is in progress, with the host and opponent actively playing.

When: After the issue is resolved, the host and opponent attempt moves elsewhere after the initial error at (1,1).

Then: Confirm that attempting moves elsewhere after the initial error results in a misleading "NOT\_YOUR\_TURN" error.

**b. Desired Behavior After Issue Resolution:**

**Test Case 4: Allow Valid Placement After Sequence**

Given: The game is in progress, with the host and opponent actively playing.

When: After the issue is resolved, the host and opponent follow the specified sequence of moves, with the opponent attempting to place "O" at (1,1).

Then: Confirm that the game accurately detects the open space at (1,1) and allows the opponent to place their mark following the specified sequence.

**Test Case 5: NOT\_YOUR\_TURN" Error Handling**

Given: The game is in progress, with the host and opponent actively playing.

When: After the issue is resolved, the host and opponent encounter the initial "PLACEMENT\_CONFLICT" error at (1,1), and the opponent attempts to move elsewhere after the initial error.

Then: Confirm that, after the issue is resolved, the game accurately handles subsequent moves elsewhere without triggering misleading "NOT\_YOUR\_TURN" errors. In this case, the game should allow valid moves and correctly indicate the turn status without the interference of misleading errors.

Task #2: Identify a Candidate Impact Set (CIS) – Before modifying the codebase to address the current issue, take time to identify a candidate impact set (i.e., the set of software entities that will be impacted by your code modification activities). Recall that software entities encompass more than source code, so consider whether portions of your project documentation and/or user guide will be impacted as well.

Although the course text provides a very formal method for identifying the CIS, I am only asking you to complete this task in a high-level manner. Simply create a list of impacted software entities and justify why you anticipate that each entity will be impacted.

**Source Code:**

* Affected code segments responsible for move validation and error handling.
* Modifications are likely required in the code that governs the validation of moves, specifically at the point of detecting conflicts and managing turn status.

**Error Messages and Logging System:**

* Messages related to "PLACEMENT\_CONFLICT", “NO\_OPPONENT” and "NOT\_YOUR\_TURN." etc.
* Changes in the error-handling logic may require adjustments or additions to error messages displayed to users, enhancing clarity and accuracy.

**Game State Management Module:**

* Modules responsible for tracking the game state, including turn status.
* To resolve the issue, adjustments might be necessary in the code managing game states to ensure accurate tracking and communication of the current state.

**User Interface (UI) Components:**

* Components displaying the game board, messages, and player interactions.
* Changes in error handling and game state management may impact how information is presented to users on the UI. Adjustments may be needed for a seamless user experience.

**Documentation:**

* Sections of the user guide explaining the game rules, error messages, and gameplay.
* Updates to the code and UI may necessitate corresponding revisions in the documentation to ensure alignment and accuracy in guiding users through the game.

**Testing Scripts and Cases:**

* Test scripts covering move validation and error scenarios.
* Modifications in the code logic may require corresponding updates in testing scripts to ensure thorough coverage of scenarios related to the identified issue.

Task #3: Perform Necessary Code Modifications – Address the current issue by modifying the codebase appropriately. Confirm that the issue is addressed using your testing plan (i.e., by manually confirming each proposed test case). Commit your modifications with an informative commit message.

All modifications were made in their own separate branch for each separate issue. Along with a commit message to that branch displaying clear transparency of the issue that was being resolved.

Task #4: Compare the Actual Impact Set (AIS) with the CIS – The actual impact set (AIS) is the set of software entities that were actually impacted during your modifications. Compare the AIS with the CIS (which was identified in Task #2) for discrepancies.

- Were any anticipated entities not actually impacted?

- Were any entities missed in the CIS (e.g., due to unanticipated ripple effects)?

**Anticipated Entities in the CIS**

* Source Code (move validation and error handling, game state management)
* Error Messages and Logging System
* User Interface (UI) Components
* Documentation
* Testing Scripts and Cases

**Hypothetical Actual Impact Set (AIS)**

Let's assume the AIS after modifications included all the anticipated entities but also revealed additional impacts not initially considered. Anticipated entities not impacted:

* It's possible that certain aspects of the **Documentation** or **Testing Scripts and Cases** were anticipated to need more substantial updates than they did.

For example, if the error messages remained largely the same or became more intuitive, less documentation update might have been required.

* The **Error Messages and Logging System** might not have needed as extensive modifications if existing messages were repurposed or found to already be clear enough once the underlying logic was corrected.

**B) Were any entities missed in the CIS?**

* **Performance Metrics:** Modifying game logic and error handling could have unexpected impacts on performance. This includes the game's responsiveness and the server's load, if applicable. This entity is often overlooked in planning phases.
* **Database Schema:** For games that track player progress, and scores, or have an account system, changes in game logic might necessitate modifications to how data is structured or stored.
* **Third-party Integrations:** If the game integrates with third-party services for features like social media sharing, leaderboards, or authentication, changes in game logic or UI could impact these integrations.
* **Accessibility Features:** Modifications to the UI and game state management could affect accessibility features, such as screen reader compatibility or keyboard navigation, which might not have been initially considered in the CIS.

Task #5: Reflect on how the system architecture affected this process – Irrespective of the Task #4 outcome, reflect on how the system architecture influenced the modifications needed to address the current issue.

- Did certain architectural elements limit the scope of necessary change (e.g., to a specific software entity or set of software entities)?

- Could architectural elements be modified to better avoid ripple effects? If so, weigh the pros and cons of such a change.

The architecture delineated clear boundaries between the front end and back end components. This separation allowed developers to focus on specific layers when addressing issues, minimizing the scope of necessary changes. For example, when resolving the reversed turn order issue, modifications primarily focused on the back end logic responsible for managing game states and turn order.

As for the ripple effect current architecture provided a modular structure, which helped in avoiding extensive ripple effects. Each component had well-defined responsibilities, facilitating isolated changes without affecting unrelated parts of the system.

If I was to suggest an improvement it would be to further separate each component into smaller cohesive modules possibly in the index.js however at the same time doing this could increase the complexity of the code base and a simple game like tic-tack-toe might not require this level of abstraction.