UNIVERSITY OF NEW BRUNSWICK
FACULTY OF ENGINEERING

SWE4203

Milestone 2

Testing, Impact Analysis, and Code Modifications

FR01B

Submitted: 2023-02-22

Members

Brendan Doucette - #3694084

Michael Scott - #3683882

Intro:

The following questions are divided into three parts, representing the three different issues we have chosen to fix.

1) Corresponds to Corrective issue 7 from the issue log: “The code to check if a game is won only checks one diagonal option twice, (0,0) (1,1) (2,2) and does not check the other, (0,2) (1,1) (2,0). As such, a diagonal victory will never be found for the one direction.”
2) Corresponds to Corrective issues 1 and 2 from the issue log: “"Your Move" "Opponent Move" Header does not always reflect who's move it is, or the proper state of the game.”, and “Header after game completion specifies "No Winner for this Game", even if a player has won.”
3) Corresponds to perfective issue 1 from the issue log: “Error messages are not user friendly, should not specify API endpoint for code related issues. Should state an easily understandable error message such as "Placement Conflict" or "It is not your turn."”

Q1

1)

1.

Given: Given a game board after the opponent player has taken their turn.

When: The player has placed the final of three markers in a diagonal line on the board.

Then: The player is awarded a win in the game.

2.

Given: Given a game board after the opponent player has taken their turn.

When: The player has placed the final of three markers in a horizontal line on the board.

Then: The player is awarded a win in the game.

3.

Given: Given a game board after the opponent player has taken their turn.

When: The player has placed the final of three markers in a vertical line on the board.

Then: The player is awarded a win in the game.

4.

Given: Given a game board after the opponent player has taken their turn.

When: The player has placed a marker on the board, that does not complete a line of three in any of the vertical, horizontal, or diagonal directions.

Then: The player is not awarded a win in the game, and the opponent gets their turn.

2)

1.

Given: A game board, where the opponents player has taken their turn.

When: It is the host player's turn.

Then: The header above the host player's board should display "Your Move" and the header above the opponent player's board should display "Opponent's Move"

2.

Given: A game board, where the host player has taken their turn.

When: It is the opponent player's turn.

Then: The header above the host player's board should display "Opponent's Move" and the header above the opponent player's board should display "Your Move"

3.

Given: A game board where the state is nearing completion for the host.

When: The host places a marker which completes a line of three of their markers in any direction (Horizontal, vertical, diagonal).

Then: The header above the host player's board should display "You Won!" and the header above the opponent player's board should display "Better Luck Next Time!"

4.

Given: A game board where the state is nearing completion for the opponent.

When: The opponent places a marker which completes a line of three of their markers in any direction (Horizontal, vertical, diagonal).

Then: The header above the host player's board should display "Better Luck Next Time!" and the header above the opponent player's board should display "You Won!"

5.

Given: A game board with eight markers placed, where the last marker to be placed can not win the game.

When: Either player places their marker in this final position

Then: The header above both player's board should state "No Winner for this Game.".

3)

1.

Given: A game board with markers placed by both players.

When: A player tries to place a marker on a non-empty position (Containing either X or O)

Then: The game displays an error message stating: “Placement Conflict: There is already a marker here, pick another position.” This message should disappear after five seconds, or after the player picks a non-conflicting position on the board.

2.

Given: A game where it is not the current player’s turn

When: This player attempts to place a marker

Then: The game displays an error message stating: “It is not your turn, wait until the opponent makes their turn.”. This message should disappear after five seconds, or after the other player makes their move.

3.

Given: A game that has been completed, by either victory for either player or a stalemate.

When: A player tries to place another marker

Then: The game displays an error message stating: “The current game has been completed. Please start another game to play again.”. This message should disappear after five seconds, or after the player starts a new game.

4.

Given: A board, where a game has not started yet.

When: A Player tries to place a marker on the board

Then: The game displays an error message stating: “The game has not yet started!”. This message should disappear after five seconds, of after the player starts a game.

5.

Given: A board, where a game has been started by the host, but an opponent has not yet joined.

When: The host tries to place a marker on the board

Then: The game displays an error message stating: “No Opponent has joined yet!”. This message should disappear after five seconds, of after an opponent joins the game.

Q2

For all tasks, “current-issues.md” will need to be updated to reflect that the fix has been completed. This will be done by applying a strikethrough to completed issues, to indicate that they have been resolved.

1)

CIS

- The only class we expect to modify to resolve this issue is the “Game.java” class. The issue is likely stemming from the “checkWon” method, where the logic to check if a player has one, does not account for diagonal movement properly, and has one check which is being run twice, just in reverse directions. The method checkWon, is private and only called within Game.java, so other classes are not likely to be affected by any changes to it. The code documentation and user guide to not describe the specific algorithm used to check if the game has been won, as such they will not need to be updated upon fixing the algorithm.

2)

CIS

- The code documentation and user-guide will be unchanged, as they are not concerned with the algorithms used to display the status. The status issue is down to a small code error, which is not described in either the guide or documentation.
- We expect to have to change Game.java and GameManager.java.
    - In Game.java, no winner is assigned as game completion is a catch-all statement for either party winning or a stalemate. As such, it will likely need to be expanded to properly indicate who has won the game.
    - This change will be reflected in GameManager.java, by updating the API response to include a winner.

3)

CIS

- The only class we expect to change in this issue is index.js. Index.js properly receives error states from the server, but displays them in a non-user friendly format, including the URL and the placeholder error names. These should be expanded to be more user-friendly, and set on a timer to disappear after five seconds, so they do not remain on the board for the entirety of the game. To do this, the following methods will need to be updated to reflect these changes
    - createSource
    - makePlay
    - get

Q4

1)

AIS

- In the process of fixing this error, we updated the checkWon method to add in the check for the other diagonal direction as mentioned in the CIS. However, we also had to make a change to the “Play” method in the “Game.java” class. On line 94 in the play() method of Game, there was a line stating “this.game[y][y] = State.O;”, this led to a variety of errors due to O being misplaced in the board. We corrected this issue by replacing that code with “this.game[x][y] = State.O;”, similar to the code on line 92 for an X move. This change inadvertently fixed a few other issues from the log, namely corrective issues 4, 5, and 6. These issues all stemmed from this line, where O was being misplaced on the board, and resolving this issue to check the diagonal has resolved those issues.

2)

AIS

- Game.java
    - In Game.java, no winner was assigned in the responses. It would simply indicate that the game was over, and as such the winners were never assigned properly. This was fixed by adding to checkFinished, to add new PlayResult states for Host and Opponent wins, rather than a catch-all game finished. Game_finished is now used to indicate a stalemate, while a player win and opponent win are assigned their own values.
    - Alongside this, the message created for the opponent to receive in index.js did not include a “winner” attribute, which was expected in index.js. The message was also being sent before the check was run to see if the game is over. Because of this, the winner was never properly assigned to the opponent player, and they were never indicated that the game was finished.
- GameManager.java
    - In GameManager.java, we reflected the changes from Game.java by adding to the response message an attribute for the winner, being either “HOST”, “OPPONENT”, or “NONE”. This ensures that the player who made the move is indicated as to who the winner of the game was to properly display the message, while the message changed in Game.java ensures the other player can see who the winner was.
- The combination of the changes in GameManager.java and Game.java resolved Corrective issues 1 and 2, as the player’s headers now properly reflect who’s move it currently is, as well as properly indicating a game’s completion, whether that be a victory for either player, or a stalemate.
- Unexpected changes
    - The order of the messages being sent in Game.java was an unexpected change that we did not anticipate in the CIS.
    - In GameManager.java, we did not anticipate the new PlayResults for victory to impact the response types. However, the 400 response was triggered because it was checking if the result was either Finished or Not Finished and did not account for a game win. As

such, we had to reflect this by adding in the new states to this statement, to ensure that they are returned and not counted as errors.

3)

AIS

- All of the changes described in the CIS were reflected here, there were no unanticipated changes made to resolve this issue. To simplify each methods error display, we declared a new method “showErrorAlert” which allows for errors to be displayed with a given message, and a given timeout time. Allowing it to disappear after the allotted time (In our case 5 seconds for each error).

Q5

To implement the changes as described above, there was generally not a large architectural problem. However, for change 2 there was a high level of coupling between Game and GameManager. This could be a frequent problem between other classes in the system, as changing code in one class often requires the other class to reflect that change. For example, adding new PlayResults to indicate game victories in Game.java caused GameManager.java to return error codes, even though the new conditions should not have been errors. It was also required to update GameManager.java after Game.java, ideally updating one of them should be sufficient to resolve issues in the system. This could be improved by implementing an architectural design pattern such as MVC to implement a more consistent separation of concerns. For example, GameManager is tasked with a variety of concepts, including managing the state of the game as well as sending and receiving API requests. This should be separated into different controllers, allowing for the modification of one to not impact the other greatly, allowing future maintenance in the system to be completed more easily. Especially when it comes to identifying the CIS for a specific issue, as with the current system it is often difficult to determine where certain actions are being performed, making the process longer than it needs to be, and involve more classes than it should. However, implementing architectural patterns such as MVC would require major refactoring of the code, as well as requiring a larger number of classes overall. This will take time to implement, as well as needing to re-test to ensure that the requirements of the system still pass under the new architecture.