UNIVERSITY OF NEW BRUNSWICK

FACULTY OF ENGINEERING

SWE4203 - Project Milestone 2

Group #2 - Matthew Collett, Eric Cuenat, Sam Keays

GitHub: https://github.com/matthew-collett/SWE4203

Professor: Dr. Aaron Tabor

Date submitted: March 1st, 2024

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:

a) Identify the current issue (i.e., at least one proposed test should fail)

- **Implement "Dark Mode" feature:**
  - Users want to be able to toggle a dark mode for low lighting environments.
    - **(Currently failing)** Given that a game is running, if the user would like to enable/disable dark, then he should be able to hit a button to switch between dark/light mode setting. Currently this feature does not exist.
- **Error Message does not clear after reset and & after valid move:**
  - Users are seeing old error messages even after hitting reset or after placing a valid move, this leads to confusion on if the game is running or if it has crashed.
    - **(Currently Failling)** Given that an error occurs, when the error is resolved then the error message should be cleared instead of seeing the old error message.
- **Fix no way to win game:**
  - Currently it is impossible to win a game, when a user wins, the message "No winner for this game" pops up when there should be a winner.
    - **(Currently Failling)** Given that a player has won a game, when the player places the third symbol in a row, then he should see a "you won" message instead of "No winner for this game" message.

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).

- **Implement "Dark Mode" feature:**
  - **(Must pass)** Given that a user is running the game in light mode, when a player toggles the dark mode toggle, then the UI should flip to a dark mode setting
  - **(Must pass)** Given that a user is running the game in dark mode, when a player toggles the light mode toggle, then the UI should flip to a light mode setting.
  - **(Must pass)** Given that a user is running the game, anytime the game is running, there should be a toggle button that switches between the dark & light mode setting.
  - **(Must fail)** Given that a user is running the game, when a player toggle dark/light mode, then UI should maintain the same dark/light mode setting.
  - **(Must fail)** Given that a user is running the game, anytime a game is being played, then the user should not be able to switch between dark/lightt mode setting.

- **Error message does not clear after reset and after valid move:**
    - **(Must pass)** Given that a game runs into an error, when this error is resolved then the error message should be cleared
    - **(Must pass)** Given that a game runs into an error, when the error occurs, then the error message should be shown until it is resolved.
    - **(Must pass)** Given that there is an error message shown to the user, when the user hits the “reset” button then the error message should be cleared if the reset has resolved the error.
    - **(Must fail)** Given that a game runs into an error, after the error message is shown to the user, then the game should clear the error message even if the error has not been resolved.
- **`Fix no way to win game:**
    - **(Must pass)** Given that a player won a game, when he places the third symbol in a row, then the game should conclude and a win message should be shown
    - **(Must pass)** Given that a player lost a game, when his opponent places the third symbol in a row, then the game should conclude and a loss message should be shown.
    - **(Must pass)** Given that there is no winner to a game, when the last player places the last symbol, then the should conclude and a no winner message should be shown.
    - **(Must fail)** Given that that there is no winner to a game, when the last player places the last symbol, then a win message should be shown to both players.

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.

- **Error message does not clear after reset and after valid move:**
    - **Source code:** A change will have to be made to the source to remove the error message when there should be no error message. We anticipate this change to be in the get() method in the index.js file after we check if an error occurred.
    - **Test Cases:** We will have to add test cases to ensure the error messages were removed.
- **No winners when there should be a winner:**
    - **Source code:** A change will be made to src/GameManager.java in the `void move(HttpExchange exchange)` function to add the winner to the response. The winner will also need to be added to the Game message in the

src/Game.java file within the PlayResult play(Player player, int x, int y) function.
    - **Test Cases:** We will have to add test cases to ensure there is a winner and loser display when the game ends.
    - **API Endpoints:** Endpoints are affected since we are adding a value into the response in order to know who the winner/loser is.
- **Implement “Dark Mode” feature:**
    - **Source code:** This will affect `src/static/index.js` since we need to include a toggle logic for our application to know what colour scheme to display. This will also affect `src/static/index.css` to incorporate the styling of the application. We will also have to make changes to `src/static/index.html` in order to add a toggle button to the screen.
    - **User Interface:** We will have to add a button on the screen if the user would like to enable dark mode. This will affect the color of the screen.
    - **Test Cases:** A test case will need to be implemented for this new feature to ensure it is working.

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)?

- **Error message does not clear after reset and after valid move:**
    - **Source code:** A change was made in the source code in the index.js file in the get() and reset(). It was anticipated in the CIS that there would be changes to the get() method but not the reset() method.
    - **Test cases:** test cases were added to ensure the error messages were displayed at the correct time. This was anticipated in the CIS.
- **No winners when there should be a winner:**
    - **Source code:** All changes that were made were valid changes. During the refactor, it was also necessary to move the `this.checkFinished();` logic before we sent the game message. This was another bug that was causing the game to have no winner since the game finished was set to false in the response when it should have been true.
    - **Test Cases:** Test cases were added to ensure that the winner and loser would get a proper response when the game ended. This was anticipated in the CIS.
    - **API Endpoints:** Endpoints were changed as planned and introduced no errors. This was anticipated in the CIS.
- **Implement “Dark Mode” feature:**
    - **Source code:** This affected src/static/index.css as we needed to incorporate the styling of the application, it also affected src/static/index.html as we needed to add a toggle button, and it affected src/static/index.js as we

needed to include a toggle logic to our application to know what color scheme to display. This was anticipated in the CIS.

- **User Interface:** We added a button to the screen that when the user presses it, it will change the color of the screen and enable dark mode. This was anticipated in the CIS.
- **Test Cases:** Test cases were added to ensure the user can change the color of the screen by clicking the enable dark mode button. This was anticipated in the CIS.
- **User Guide:** We had to update the user guide incase the user would like to enable dark mode. This was not anticipated 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 creational design patterns used in the architecture of the system restricts us a lot for small changes. A lot of the components are highly coupled and violate the Single Responsibility rule from the SOLID principles. A lot of the functions are doing multiple things at the same time leading to code duplication and unexpected ripple effects.
Introducing interfaces or abstract classes could help in decoupling components, allowing for more isolated changes and reducing the risk of ripple effects. However, while they can improve flexibility and maintainability, they might also introduce complexity and require additional time for development and testing.

The architecture also violates the Open/Closed principle, making new changes involving extending functions leading to confusion and duplicate code again. We can use polymorphism to extend behaviour, by incorporating the decorator or strategy design pattern.

Trade-offs include increased complexity and losing time to rewrite code that has the same functionality. However, this will make the system much more adaptable to change allowing us to introduce new features with less risk of introducing bugs