Group: Badger

Members: Zachary Chan, Kunlong He, Curtis Huang, Kyle Mollard, Angela Yung

Refactoring

By the end of phase 3, we improved the overall quality and maintainability of our code by addressing code smells that affected our project. Code smells are indicators of issues that can lead to higher susceptibility to bugs which is a situation we want to avoid at all costs. Below we discuss and provide examples of three different code smells we found, what issues the code smell created, our refactoring techniques, and how refactoring improved the code.

Dispensables: Duplicate Code

Here is an example of duplicate code that was refactored: https://github.sfu.ca/aya53/board-game-project/pull/11/commits/abf6a210e2d5f7db3ebfbe7f0b293db9f06586a4

We noticed a significant amount of duplicate code in the files *cheating.js* and *heat_mp.js*. Because our code base was rapidly increasing in size, large amounts of unnecessary duplicate code made the project harder to maintain and the code harder to read. The duplicate code consisted mostly of constant variables so our work-around was to remove these constants from both files and store them in a separate file called *constants*. Now, there is one file that holds all the constants that both files can access using an import statement. We chose this refactoring solution because it was easy to implement and did not contribute additional complexity to our code structure. As a result, the code is much easier to read and understand in both files, and it is easier to make changes to the constants since they are all in the same file.

Dispensables: Lazy Class

Here is an example that shows how we refactored a React component that contributed almost nothing to the overall game (note: we used components over classes in React so refactoring involved changing components instead of classes): https://github.sfu.ca/aya53/board-game-project/commit/978d83c4ed2ab24da113f7e3916be9845f7337a8

The header component in *header.jsx* was originally intended to be a placeholder for the title during the early stages of development. That was its only purpose. Now that we have improved our main menu UI, we do not need a separate component for the header. Without refactoring, the component would add to the rapidly expanding code base and reduce code maintainability. Our solution was to remove the component and file all together. However, that means we can’t display the title anymore. Fortunately, there is an easy way to print the title in the main menu using just 3 lines of code. As a result, we removed 20 lines of code from *header.jsx* dedicated to printing the title “Battleship” and added only 3 lines of code in *main_menu.jsx* to print the title. The code is now easier to understand because the title belongs in *main_menu.jsx* instead of a separate file, and we decreased the size of our code base which means less room for bugs.

Bloater: Primitive Obsession

The following is an example of how we refactored code to address over-use of primitives (specifically arrays): https://github.sfu.ca/aya53/board-game-project/commit/5c70bb559fe19b084e714a994d9a98e53568a8d5

The primitive data type that was used in this case was arrays. The problem posed by using arrays in our code was that we would retrieve more data than we needed. When transitioning to objects, we gain the advantage of easily destructuring the object by specifying only the necessary data elements. This eliminates the need to retrieve all the data even if we do not require all of it, resulting in more efficient and concise code. In essence, the use of objects over arrays addresses the code smell, enhances code readability, promotes better data access practices, and supports efficient deconstruction of data by extracting only what's needed.