# Assignment 4 - Code Review

Methods that are too long and that could benefit from being refactored:

- To address the identified code smells, we applied the Extract Method refactoring technique to reduce the size and complexity of lengthy methods and improve readability, maintainability, and modularity. For methods such as UI Draw PlayState and UI Draw PauseState, we created helper methods like drawPlayerStats(g2), displayMessage(g2), and drawPauseMessage(g2) to isolate distinct responsibilities. Similarly, for the Player Update method, we introduced methods such as isMoving(), handleMovement(), updateSprite(), checkGameStats(), and updateExitDoorState() to separate concerns and streamline the player's update logic. Each extracted method encapsulates a specific task, making the code easier to understand and test. After each refactor, we reran the test suite to ensure the program's behavior remained consistent. The updated design follows the Single Responsibility Principle and promotes better code reuse and scalability

- In UI drawPlayState we created helper methods like
    - drawPlayerStats(g2):- This method is used to display the Player Stats on the game panel.
    - displayMessage(g2) :- This method is used to display the player object interaction messages.
    - handleBonusObjects():- This method is used Handle the bonus Reward placements.

  **Commit : 32743bf**
- In UI drawPauseState we created helper methods like
    - drawPlayerStats(g2): This method is used to display the Player Stats on the game panel.

      **Commit : 32743bf**
    - drawPauseMessage(g2): This method is used to display Pause state messages.

      **Commit : 5b20b0c**
- In Player Update we created helper methods like
    - handleMovement() : Method used to check collision along with movement direction of the player

      **Commit : 22307dd**
    - checkGameStats() : Method used to check the negative points and door-unlocking logic

      **Commit : e5e4df0**

Code Duplication:

- To eliminate code duplication and improve reusability, we refactored sections of the code that repeated functionality across different components.

- In the Object classes, we centralized the image loading logic by creating a new loadImage() method in the SuperObject class, allowing all subclasses to share this functionality without duplicating the code. This adheres to the DRY (Don't Repeat Yourself) principle and simplifies future maintenance.

  **Commit - 5b8e6bf**
- We addressed the duplication of drawPlayerStats(g2) in both the PlayState and PauseState in the UI class by consolidating this method into a single reusable implementation, which can be invoked from both states. These changes enhance consistency, reduce redundancy, and make the codebase more maintainable while preserving its functionality. **Commit - 32743bf**

Unnecessary if/else or switch/case statements:

- To address the problem of excessive and unnecessary if/else or switch/case statements in the UI class for checking the game state, we introduced a Map-based dispatching mechanism. By defining a stateDrawActions map that associates each game state with its corresponding drawing method (e.g., drawMenuState, drawPlayState), we replaced the verbose conditional logic with a concise, scalable,

and dynamic approach. The new implementation in the draw method retrieves the appropriate drawing action for the current game state and invokes it via a Consumer<Graphics> interface.

- This approach improves readability, adheres to the Open/Closed Principle by enabling new states to be added without modifying existing code, and enhances maintainability by centralizing state-action mapping in one place. **Commit - 674db1e**

Badly structured project (i.e., file setup):

- In our old structure, Classes were loosely grouped into broader categories, which mixed different functionalities. For example, collision-related classes were scattered or combined under unrelated packages. UI and utility classes were not clearly separated, making it harder to locate specific functionality.
- The updated project structure organizes classes into cohesive packages based on their functionality. This makes the project easier to navigate, extend, and debug.
- This new structure follows the **Single Responsibility Principle** (SRP), where each package and class is focused on one aspect of the game, reducing coupling and enhancing cohesion. **Commit - ce5399d**

Added Exceptions:

- Image Loading Exceptions - We created image loading exceptions for player and object image resources which improved upon the original exception handling which was just using IOExceptions and printing stack traces. The custom exception handling helps provide more context and gives the path of the image resource which caused the error. **Commit - 95b94c9**
    - Player
    - UI
    - Object
- Sound Loading Exceptions - We introduced sound loading exceptions to handle errors during the loading of audio files for the game. The original implementation relied on generic exceptions (IOException, NullPointerException, etc.) which provided minimal context about the cause of the failure. The new custom exception, SoundLoadingException, offers improved handling and context by including the path or URL of the problematic audio resource. **Commit - 6be8943**
    - Sound

Classes That Are Too Large or Do Too Much:

- To improve the design and code quality, we identified that the Player class was too large and did too many things, leading to reduced maintainability. To address this, we refactored the code by creating a new PickObjectHandler class, which is now responsible for managing object interactions such as collecting clovers, carrots, and mushrooms. This refactoring reduced complexity, making the Player class more focused on player movement and state, while delegating the object interaction logic to the new class. We reran the tests after each change to ensure the program's behavior remained intact.

  **Commit - 9bc19f1**