Code Review

Brayden Sue and George Cheng

Code Smell: Unjustified use of primitives
Resolution: Replacing primitives with objects

Before this change, stationary entities were identified in the asset setter by their names, which were stored in strings. This is an unjustified use of primitives because there are only a few possible types of S_Entities. Our solution was to replace the String values with enum objects, then we ensured that all test cases still passed. By applying this refactor, we were able to increase the readability of our code by using discrete EntityTypes rather than arbitrary strings. This refactor also improves maintainability by enforcing a categorization system for each distinct object type being used in the game.

Code Smell: Low cohesion
Resolution: Moved functionality to child classes

In our Player class, when an item pickUpObject(int index) was called, the content of the tile in our 2D stationary entity array was directly accessed by its index and set to null. To increase encapsulation and cohesion, we added an interface for Destroyable objects with the method self_destruct(), then added the body of the function to the destroyable objects' superclass, S_Entity. Since all of the objects that implement Destroyable execute their self-destruction in the same way, special cases were not required for any of the classes. After moving the functionality to the child classes, we ensured that all tests still passed. This modification also improved the readability of our pickUpObject() method, as it is easier to understand that an object is being destroyed through a function call than an operation on our 2D array.

Code Smell: Duplicated code
Resolution: Deletion

In our asset setter class, we assigned the game objects to their appropriate index in our 2D array using two methods, placeObject() and setObject(). Initially, setObject() was used to add an object of a given type to the 2D array obj, while placeObject() was used to assign in-game x and y position values before adding the object to obj. Our solution to these similar methods was to combine them into one that performs both of their responsibilities. Because all objects in the game should have a position, there was no reason to have them in separate methods. After deleting placeObject() and modifying setObject(), we ensured that all tests still passed. This change improved the maintainability of our asset setter To improve the maintainability of our asset setter, we combined the functionality of the two methods into one that adds the objects to obj, complete with position values.

Code Smell: Bad/confusing variable names
Resolution: Renaming variables

Some variables in the class GamePanel had confusing names that made it unclear what they did exactly. We modified the names of some of these variables so that with one glance you knew what these variables did. This improves the maintainability of the code by reducing the time taken to relearn what those variables are used for.

Code Smell: Bad/confusing interface name
Resolution: Renaming interface

The interface that is used for objects that disappear after interaction was called Consumable. While this was appropriate for the rewards, bonuses, and arguably the hazards, we decided that Destroyable would be a more suitable and descriptive name, as the self_destruct() method can also be applied to other objects, such as doors. This change improved the readability of our code and can help to avoid future confusion about the behaviour of objects.

Code Smell: Dead code
Resolution: Deletion

All of the graphics we used for our user interface are loaded into object classes, which extend the UI_Object abstract superclass. We noticed that the draw() method in our UI_Object class was creating redundant variables by adding and subtracting the same x coordinate values for a new variable. Upon further inspection, we realised that the method is not called from anywhere in our project. The method called the Graphics2D drawImage() method on a given object, but to allow for explicitly stated UI object positions, it was not used. Because this method is not currently being called from anywhere in the package, it is dead code and was commented out as a result. After commenting out the method, we ran the tests to ensure they all passed. By acknowledging this method as dead code, the maintainability of our system was improved.