Derek Huang, 301441129
Nancy Wang, 301394970

Code Review

In this report, we explored the codebase for our app to identify code smells and ways to refactor them. We chose to analyze and refactor the classes Zombie.java, Player.java, and Sound.java. These were not only central to our game's functionality but also presented opportunities for significant improvements. Our aim was to streamline the code, making it more readable, adaptable, and robust against future changes and additions.

Code Smells

1. Long update method for Zombie class
    a. We split up the update method into smaller methods for better clarity. It was fairly simple to do since we noticed that in the method there were clear blocks of code that were doing a specific task within the update method. Thus, we split into 3 smaller methods of 'setZombieDirection', 'handleCollision', and 'updateAnimation'. Originally we had tried refactoring it into 4 smaller methods but it did not retain the same behaviour so we settled for 3 smaller methods. See commit b705698
2. Adding polymorphism to entity class (common methods from player and zombie)
    a. The Entity.java class is the superclass for both Player.java and Zombie.java, but there weren't any polymorphic methods in the entity class since we only have 2 subclasses that extend Entity. However, if we need to make more entity subclasses in the future, it will be more efficient to use polymorphism. This enhances scalability and code efficiency. See commit bd7699b.
3. Adding a null check to Sound.java
    a. A recent edge case was identified during testing, where calling the stop() method without any sound playing led to an error. To resolve this and pass our structural test cases, we've refactored the Sound.java class to include a null check for all methods, ensuring stability and error prevention in sound management. See commit d4e3f20.
4. Deleted random commented out code/useless comments
    a. Unnecessary commented-out code and redundant comments have been removed for cleaner code. Additionally, we've moved relevant comments to Javadocs and adopted self-documenting method names in Zombie.java and Player.java to improve code readability and understanding. See commit e97b11a and commit bd7699b.
5. Getting rid of comments and adding java docs in zombie class
    a. Removed all the comments and added java doc. See commit d71adb1.
6. Use the helper function to retrieve zombie images
    a. Our code has implemented the function of using the setup method to retrieve and process zombie images from resource files, meeting the need to use auxiliary functions to obtain zombie images. This approach ensures that the zombie

animations appear correctly in the game and are consistent with other graphical elements of the game. During refactor, we put the setup on top of getZombieImage so that we can better understand the code. See commit a4a1cfc.

7. Optimize Switch cases for player class
    a. We are improving the player.java code. To manage player interactions with game objects, we first swapped out the switch statement in the Player class with a hash map and method references. Next, we made the ObjectAction interface parameter-accepting, making sure that related methods got the same parameters. In order to improve the readability and maintainability of the code, we have generated comprehensive JavaDoc comments for every function. See commit 01bb00a
8. Optimize Switch cases for zombie class
    a. Optimized zombie.java, this class is responsible for managing zombie animation and behavior in the game. I used a Map instead of a switch statement to have more flexibility with the zombie's orientation and animation frames. See commit 467dcf7
9. Fix long update for player class
    a. Split the update class into small classes. See commit 7cde40c

The refactoring journey we embarked upon has led to substantial improvements in our code's structure and clarity. By breaking down the monolithic update method in the Zombie and Player class, introducing polymorphism in the Entity class, instituting null checks in Sound.java, cleaning up comments, and optimizing switch cases with hash maps and method references, we've laid a stronger foundation for our code base.

Derek Huang, 301441129
Nancy Wang, 301394970

Code Review

Maven Project Structure:

One way to improve our Maven Project structure is to use packages to organize java classes.
In order to do this:

- All the objects classes in an object package
- Player, Zombie and entity classes in a package called entities