CMPT 276
Assignment 4

List of Code Smells

1. Duplicated Logic
2. Feature Envy
3. Poorly Structured Code
4. Lack of Comments
5. Unhandled Methods
6. Long Methods
7. Poor Variable Naming
8. Large Class
9. Magic Numbers
10. Change Preventers

1) Duplicated Logic (Link)

Within some functions, such as isWall, checkCollision, and checkObjective, similar logic is repeated for iterating through objects and comparing positions between two objects.

2) Feature Envy (Link 1, Link 2)

Certain functions, such as movePlayer and moveEnemies, manipulate other objects directly. Instead of having these functions in Level.java, these were moved to each of the classes that they were manipulating, e.g. moveEnemies was moved to MovingEnemy.java.

3) Poorly Structured Code (Link)

For example, in Level.java, certain loops and if statements used nonoptimal logic to iterate through an array or as conditions for if statements.

4) Lack of Comments (Link)

Under certain complex functions, there was a lack of comments to describe how the functions worked, so comments were added for legibility.

5) Unhandled Methods (Link)

In Game.java, there were two unhandled features that were looked at specifically: keyTyped and keyReleased. They were implemented from KeyListener but had no indication as to why they were left empty and/or unused, so comments were added stating the reason.

6) Long Methods (Link)

A prime example of this is in Game.java's keyPressed function. This function attempted to check for key presses and which direction they were but also tried to handle other things, such as incrementing score, moves, etc., so this was moved to a different function, handleAfterPlayerMovement.

7) Poor Variable Naming (Link)

Vague variable names such as onDoor were changed to playerIsOnDoor as it's more descriptive and specifically is only used for the player.

8) Large Class (Link)

In GamePanel.java, it was observed that the parameter list was eight parameters long. Furthermore, a majority of these parameters could be bundled together within another class. In this example, a lot of the parameters were attributes of the Level class. This was handled by performing *Extract Class*, and simply passing a Level object as a parameter instead of a whole sequence of Level's attributes.

9 & 10) Magic Numbers, Change Preventers (Link 1, Link 2)

Across a lot of the project, magic numbers were being used to indicate things such as the number of lives a player has or indicate the location of assets. These were all moved to Values.java to centralize everything and make it easier to replace values if necessary instead of having to comb through all the files to make sure things are appropriately changed.