Assignment 3: Code Review

Names: Noah, Lionel

Part of the project that we are refactoring:

Entities folder, UI class, Keyboard, Player, AnimateEntity, Enemy, CheckCollision

Smells:

- Bad Smell 1: **Badly Structured Project**

We noticed that the project structuring was inconsistent and unorganized. For example, the entities folder consisted of animate and inanimate entities. Inanimate entities were renamed to “object” and moved into a separate folder, but animate entities remained in the entities folder and were very clustered. We decided to move all subclasses of animated entities into a new folder called “moving_objects”.

- Bad Smell 2: **Dead Code**

I found some unused parts of the code within the AnimateEntity class, such as some getters and setters. I found out this part was unused from the IDE I am using, IntelliJ Idea. It allows you to scan your code and when the code isn’t being used it makes the method's name grey, but when it is being used the name is coloured. Typically the colour of the name for methods would be yellow and for variables, the colour would be purple. For getters, I removed the get_spriteCnt and for the setters, I removed set_spriteCnt and the set_spriteNum.

- Bad Smell 3: **Unencapsulated Class in UI**

In the UI class, we realized that we could increase the security of the class by encapsulating all its necessary fields. We followed the principle of “encapsulate what varies” by making public fields into private fields and implementing the appropriate getters and setters. Also, fields that shouldn’t change were appended with “final”.

- Bad Smell 4: **Convoluted/Confusing Methods**

In the AnimateEntity and Player class, some methods were long and confusing with inappropriate documentation. To fix this problem, there were multiple changes made:

- Shortened and decreased some conditional and switch statements
- Decomposed large methods into smaller ones

One example is the update method, which had lots of conditionals that clustered up the method. We shortened and extracted parts of the conditionals into new methods, making the class and documentation more clear.

- Bad Smell 5: **Unnecessary If/Else or Switch/Case Statements**

In the KeyBoard class, I first noticed there were a lot of unnecessary or statements within the if statement that checked whether we were in a state that used the arrow command. I refactored this by changing the if statement to check if it was greater than the sim.pauseState (2) and only then would the nested code below run. This worked because the only 2 states that didn't use the arrow command were pauseState and playGameState which were just variables for 1 & 2 and all the states that used it were variables for 3 and above. After I realized that I could also remove a bunch of if statements because a lot of them were doing the same thing, such as quitting the app and entering the playGameState. I refactored this code by just adding or statements to the existing if statements to check for the other states that I wanted to add to it. Lastly, I just updated some previous comments to explain the commands better and also added new comments for the new implementations.

- Bad Smell 6: **Duplicate Code**

For AnimateEntity and its subclasses (Player and Enemy), we noticed that there were lots of duplicate code, such as moveEnemy and movePlayer. To fix this problem, we changed the moveEntity method in the animateEntity to be compatible with both the player and enemy classes. This change allowed player, enemy, and future animateEntities classes to call the moveEntity method in the superclass instead of implementing their own move method.

- Bad Smell 7: **More unencapsulated classes**

This was written in a separate section as it consisted of lots of small changes in multiple classes. After refactoring the UI class to be more encapsulated, we noticed that there were some security issues in multiple other classes that could be encapsulated. This includes the CheckCollision, AssetCreator, and Sound classes. We made inappropriate public classes private and implemented appropriate getters and setters.