**Feature Envy**

in the obstacle class we had calculations done for the obstacles that are spikes. Spikes should be its own class as Spikes should be a subclass of obstacle. So, I used the extract method and took out anything that had to do with spike and made it its own class, as well as made it a subclass of the obstacle class. This is more in line with object-oriented design, as a spike is an obstacle. The testing done is in Obstacle Test as this class test uses through spike since obstacle will never be called directly. It will always be called by the subclass of the Obstacle class. So, no tests are done directly through obstacle. What’s checked in the test is only the hit detection in the spike class. The other methods in the class don’t seem to need testing since they are simple enough pieces of code that testing would be a waste such as a getter setter and a random generator that picks a starting point for the spikes. The code after the refactoring is in a much better format, as said before it is more in line with object-oriented design now. This means that in the case of expansions that could be done it will be much more read able as well as easier to make more obstacles.

**Inappropriate intimacy X 3**

There were many cases of inappropriate intimacy through out the code. The first case was in the GUI class. For the methods initialize and move, initialize turns private since its only used in the GUI class as a helper function no other class needs to use it so its private. Move is called in the main function to start the game which means we cannot have it private, but it should still be protected since no classes in the logic classes need access to it. Next case we have is in the world class this simply changes the function paint component to protected since its only called by GUI and nothing in logic. Therefore, protected is the right approach. This was all done in commit 1987711fb73f9eb048c03182e35bc8ee81ba1d2c. in commit 59cc31f5414a50f70cdc5d68a0a2f89de5455272, we see in class obstacle we fix one instance of inappropriate intimacy by changing obstacle interaction to protected since it will only be called by its children. Its simply java practise to keep permissions as private as possible and increase as necessary only the functions that absolutely must be public should. As you don’t want functions calling function they shouldn’t be.

**Lazy class**

In the scores and player class we see an instance of a lazy class. We have two main issues or “bad smells”. One is its very minimal, in scores all we simply do is read and write to a file. Very minimalistic point of details and does not really prove its worth having around. The other main issue with it is in terms of object-oriented design it doesn’t make sense. Scores extends player, score is not a player rather a player has a score. This means that score should be inside of player. So, I used collapsed hierarchy to fix the issue. Now in terms of actual refactoring its mainly making sure then whenever there was a reference towards scores was to switch to player. Aside from that the testing that needs to be done is really making sure the player class has not gained any bugs unnecessarily, since the code is simply reading and writing to a file , and the fact that its only writing a single integer to the file there is not much wrong that can go wrong. All other functions are tested in the player class to ensure the added code didn’t cause any issues.

**Form Template method**

The obstacle class has an obstacle interaction method and before it was tailor made for simply the spike but now if another obstacle is created, the obstacle class can be used once again for another obstacle.

**Shotgun surgery**