SER 316

Edgar Hakobyan

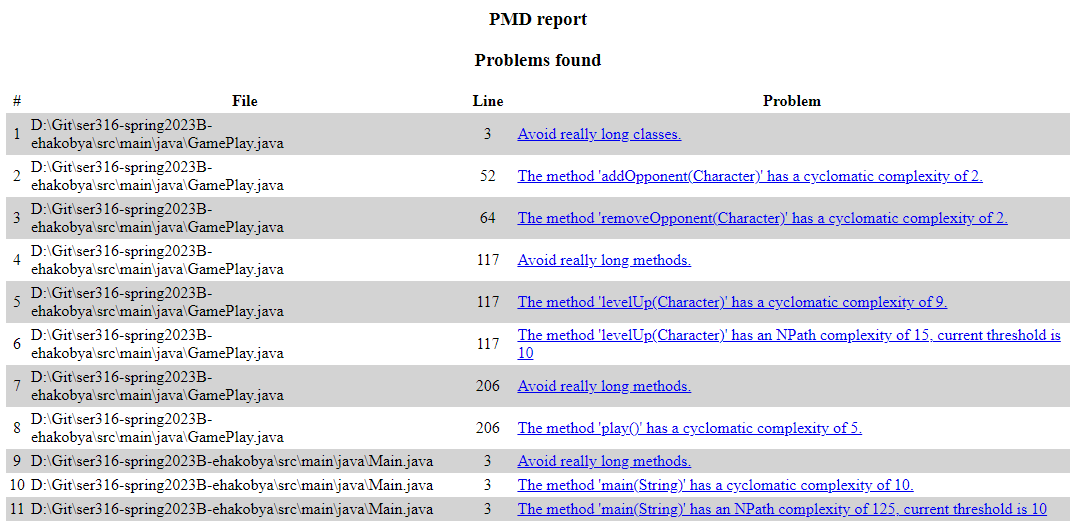
04/27/2023

GitHub:

Assignment 6

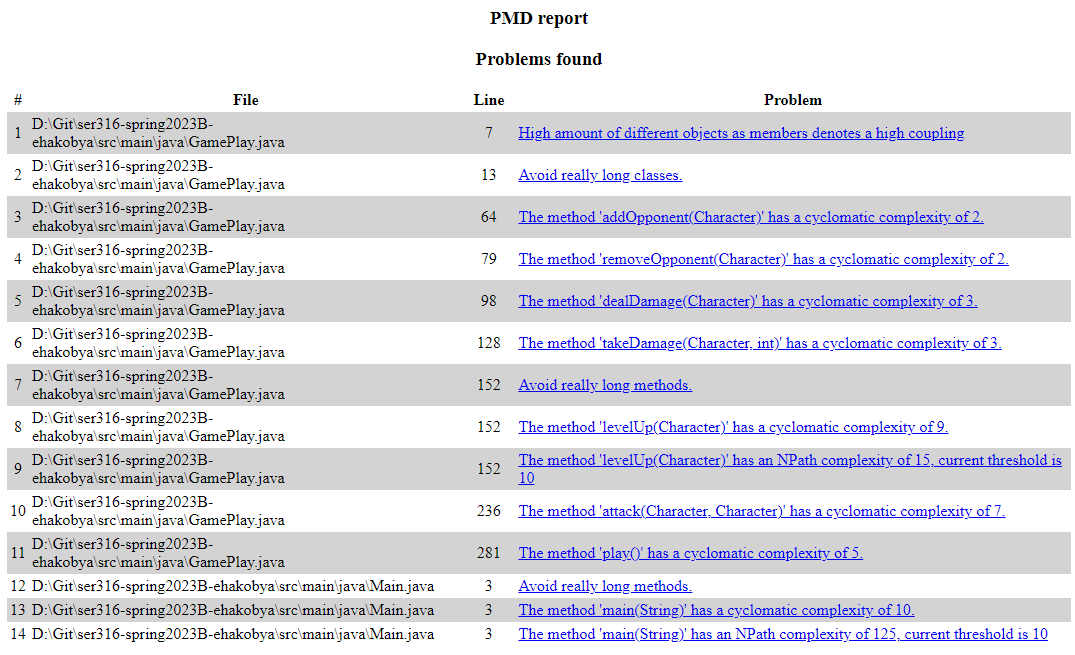
**Task 1**

Upon examining the ruleset, I noticed that the ruleset defines a cyclomatic check with level 2. Level two is a very low level purposely set to show something in the report (google says anything below 4 is good). Also, I see that there are also Coupling, Class Length, Method Length, Path Complexity and Loose Coupling checks in the ruleset. All of these “properties” have a certain threshold associated with them which are also reflected in the report. For example threshold for Class Length is set to 100; therefore, error #1 states that it is really long class.

****

**Task 2**

The number of problems has increased from task 1. This is due to the fact that dealDamage and takeDamage where not implemented int the blackbox testing phase. Same exact problems persist from task 1 to task 2 with the addition of new problems in regards to the methods mentioned previously. Based on this information, we can say that there are no improvements in terms of problem solving.



**Task 3.1: Adaptive Code**

One way we could make code adaptive is using the strategy pattern for the level-up method. The purpose of the strategy pattern is to define multiple algorithms for a specific task, and let the application decide which one to use during run time based on some user interaction or code logic. This pattern works well for level Up method because each character type levels up differently (gains different attributes). Therefore, their level-up methods are slightly different. Hence, using the strategy pattern to define a family of algorithms and use the correct one based on which class is leveling up during run time.

**Task 3.2: Duplicate Code**

Only place that has duplicate code is the **Main** class. There are 3 games that are happening in the main and the code for it is duplicated. We could combine all three into a single for loop with 3 iterations and eliminate code duplication.

**Task 4: Code Smells**

Here are some rules that we could add to the ruleset to help us find code smells.

**Comment Size:** Can help to reduce overly commented code. According to fowler, overly commented code is usually bad code.

**Data Class:** This can be used to identify classes that are only used for storing data such as the character classes.

**Excessive Parameter List:** We could set a threshold value to limit the number of parameters that are passed to a method. It is better practice to replace long parameters with an object that contains getter methods to access the data.

**Task 5: Final check**

I believe it is not possible to have something be perfect. There are always going to be flaws of some sort. However, with refactoring we could greatly minimize the flaws and get the implementation as close as it gets to perfect.

Some methods in the program have high cyclomatic complexity. I would try to simplify those methods. Also, some methods have very high NPath complexity, which means there are many different ways the code could execute. This is a little bit of an issue because it makes code less predictable and harder to test.