Analysis of the group review

We greatly appreciate the feedback of the other group on our project. There were some useful tips in it and we'll use them to improve the quality of our game even further.
However, at some points we disagree with the other group, mostly because we cannot do anything about it. These points at which we disagree with the other group are marked **bold.**

The source code looks very well designed. A wide variety of well implemented design patterns have been used like Strategy design patterns , Observer Design pattern and Factory Design Pattern. The usage of Enums in this project and the usage of a constant file is a very good in the project. The usage of asserts could have been better as they are used sparsely in the project.	We'll pay more attention to check the class invariants (by using asserts)
Some of the things that are not good with this game is that it cannot be run out of the box on most of our team's pc's	**This is because pom.xml was not included by the TA's**
Also some of the checkstyle rules used are not clearly explained in the source code, for example the max length of a line is 200 characters instead of the normal 100 characters	We'll document the reason for changing the CheckStyle rules
There are 53 Checkstyle errors in total but these are easily solved as they all are errors indicating the file does not end with a new line. PMD gives also some easily solved errors like unused variables , which do not necessarily reduce the code quality	We'll solve this week all CheckStyle/PMD/FindBugs errors
There are also some unnecessary classes which could be removed. For example the package-info.java class in every package that does not contain any real classes but just the name of the package could be deleted.	**The package-info.java cannot be deleted, because this contains the javadoc of the package. If we would delete this, CheckStyle would give a warning**
The project pom file does not contain a specific setting to build the game, so multiple steps have to be taken to build the game and be able to run it. Also no option to make a JAR file is implemented in the pom file	**This is because pom.xml was not included by the TA's**
A travis configuration file is missing and Travis is not working without.	**This is because travis.yml was not included by the TA's**
Also Findbugs has not been used in the project , at least no report for Findbugs is generated.	**FindBugs is included and the report is generated a great many times, but this was not included by the TA's**
In maven a test report is not generated , this reduces the ability of product owners to determine in an easy way if a projects is stable enough for deployment	**In Maven a test report is generated, but this was not included by the TA's**
The branch coverage of the project itself is 7.8% which comes because of the unnecessary files in mentioned earlier but also because the code is tested very poorly. The developers had tried to implement integration tests with Cucumber but it has failed drastically as they have used powerMock instead of the normal Mockito.	**This is because the only the non-Cucumber tests were run. Without PowerMock it is impossible to tests private methods and classes. Cucumber has little to do with PowerMock**
The code follows the language conventions completely. Variables and methods have the right use of first having a lowercase letter and for every new word that is added to make the whole name uppercase letters. Final variables are in full uppercase. Classes have the correct use of CamelCase. Also proper use of indentation for nested code.	-
Naming is perfect. Really no room for improvement. Variable names are short and to the point. Same goes for methods. Classes also have short names. They start with an I when it's an Interface and with an A when it's an abstract class. This really improves the readability of the	-

program as a whole. The structure of the folder is also very good; clear single-word subfolders containing all classes.	
Again, 10/10 because all methods have comments. They are all in the same style and with good punctuation. All variables have comments as well. Long methods have single line comments making it very clear to see what happens in the code. Slightly more complicated methods have longer comments; kind of stories to explain what happens. In these cases it does not matter that the comments are not short; it is needed to fully understand it all	-
We think the game is nearly finished already, so no actual improvements are needed in terms of design patterns being applied in the code. Also the readability of the code is very good, so no enhancements needed there.	-
The code quality of the game could be increased by numerous things. 1. Testing coverage should be increased , at the moment a branch coverage of 7.8% is implemented , this has to be increased to 75%.	If the tests are run with Cucumber and Powermock, the test coverage is 50% instead of 7.8% . However, they have a good point that the testing coverage is too low and we'll set the increasement of the test coverage high on our priority list
2. During testing the PowerMocks should be reduced as this is bad practice	**?** **Only programs that modify the Java bytecode (like PowerMock) are able to test private methods.** **2 alternatives: increase the scope (very bad practice) / don't test the private code at all (very bad practice) / test the private code indirectly (practically impossible because every single constructor is private)**
3. Maven test reports should be generated for the project.	**These were not included by the TA's**
4. If a Travis config file is not present , it should be made.	**This was not included by the TA's**
5. Some of the powerUps are not fully implemented as we saw the code for it but it never came back in the game.	Good point, last week these were implemented
6. Findbugs reports should be added to the maven site	**The whole pom.xml (including FindBugs) was not included by the TA's.**