Assignment 5- Group 8008- SEM

Joost Verbraeken 4475208
Eric Cornelissen 4447859
Nick Winnubst 4145747
Cornel de Vroomen 4488628
Michael Tran 4499638

Exercise 1 – Anonymous Peer Suggestions

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 points at which we disagree have mainly to do with files (pom.xml, travis.yml, generated files) that were not included in zip they got apparently, but that are present in our project.

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 in the zip apparently. If it was included, the project can be run on every pc with a clean installation. If this was not the case, Travis would have failed to compile and/or run the tests.**
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	**The package-info.java cannot be deleted, because this contains the javadoc of the package. If we would delete this, then there is no documentation of the package**

classes but just the name of the package could be deleted.	**available! (resulting justly in a CheckStyle 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	**? We can run the game directly, without any additional steps, just by running main.** However, the point that is made about the jar file is important. Last time including the JAR option didn't work because of issues with the way we read files, but we'll try again to fix this.
A travis configuration file is missing and Travis is not working without.	**This is because travis.yml was not included in the zip**
Also Findbugs has not been used in the project , at least no report for Findbugs is generated.	**FindBugs is included and we generated the report multiple times, but both were not included in the zip (apparently)**
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	**With Maven a test report was generated quite a few times, but these reports were not included in the zip**
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.	**The maven test report was not included in the zip. However, using "mvn site" it can be generated very easily (if our pom.xml is available)**
4. If a Travis config file is not present , it should be made.	**This was probably not included in the zip**
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	We'll check pom.xml again and make sure the FindBugs report is generated perfectly next time

Exercise 2- Software Metrics

1 Summary

This report was made with the purpose of analysing and explaining faults and corrections for the Doodle Jump project as a part of the Software Engineering Methods course at TU Delft during Q1 2016/2017.

2 Analysis

The initial results of the analysis using inCode can be found in the source of the Github repository and named "DoodleJumpSoftwareMetrics.result". Looking over the automated analysis we find faults divided into 3 categories:

• Data Classes (3x)
• Tradition Breakers (2x)
• Data Clumps (15x)

Each of these categories will be discussed separately, and if warranted corrected. It has to be noted that the most severe violations (severity 2 and 3 compared to 1) are all located in the completely corrected set of Data Clumps.

3 Data Classes

In the initial analysis, it is determined that the project contains three classes that expose a lot of their internal structure, also known as data classes. The classes identified as such are:

• Sprite.class
• DefaultProgressionObserver.class
• SaveFile.class

These are correctly identified as such. In this case, being a data class is actually their intended use. This is done in accordance with splitting out responsibilities. Concluding, it can be stated that while these classes are rightfully identified as being Data Classes, due to their intended nature, these classes are in fact not flawed.

4 Tradition Breakers

In the initial analysis, it is determined that the project contains two classes that wrongfully implement inheritance due to either leaving methods as NOPs or by not showing all of the inherited interface. These two classes are:

• StartScreenDoodle.class
• ServiceLocator.class

Firstly, the StartScreenDoodle.class is correctly identified as supposedly wrongfully overwriten certain inherited methods with NOPs. This might cause issues if one accidentally overwrites a method with actual use. The methods in this case, are all related to the controls of the doodle. Considering the fact that you don't want the StartScreenDoodle to be controllable, it is justified to overwrite the methods relating to these controls with NOPs.
Secondly, the ServiceLocator.class is correctly identified as not showing all of the methods of it's inherited interface. This can cause problems if the super class is an abstract class or uses NOPs for

these methods. In this case, the super class implements all methods and this extension is simply to separate a service locator with and without audio. This issue could be resolved by implementing all missing methods, but this would cause unnecessary duplicate code. This causes the conclusion that while the conclusion of the analysis is true, in the current implementation it can be ignored due to the fact that any alternative cause other problems.

5 Data Clumps

In the initial analysis, it is determined that the project contains 15 methods that contain a Data Clump in the form of containing too many parameters. This can cause issues due to the fact that when programming it can cause trouble due to the amount of parameters that need to be passed. This causes an, in many cases, unwarranted complexity for the coder. An approach to improve this is by combining different parameters together into a single parameter. This approach has been applied to the 15 affected methods (and a lot of other methods that are affected by the widespread use in the program of the conventions that caused these issues). This meant the coupling of pairs of integers into pairs and the conjunction of sets of sprites.

Exercise 3- Teaming up

The CRC cards for this implementation:

Doodle

`Superclass: AGameObject`

Purposes	Collaborators
Adds experience to its attribute	World

World

Purposes	Collaborators
Simulate the game	KillScreen
Check for collisions	Doodle
End game when doodle dies	ProgressionManager

ProgressionManager

Purposes	Collaborators
Keep the total experience	Ranks (Enum)
Keep the rank	ChooseModesScreen
Modify and retrieve the rank and experience	World

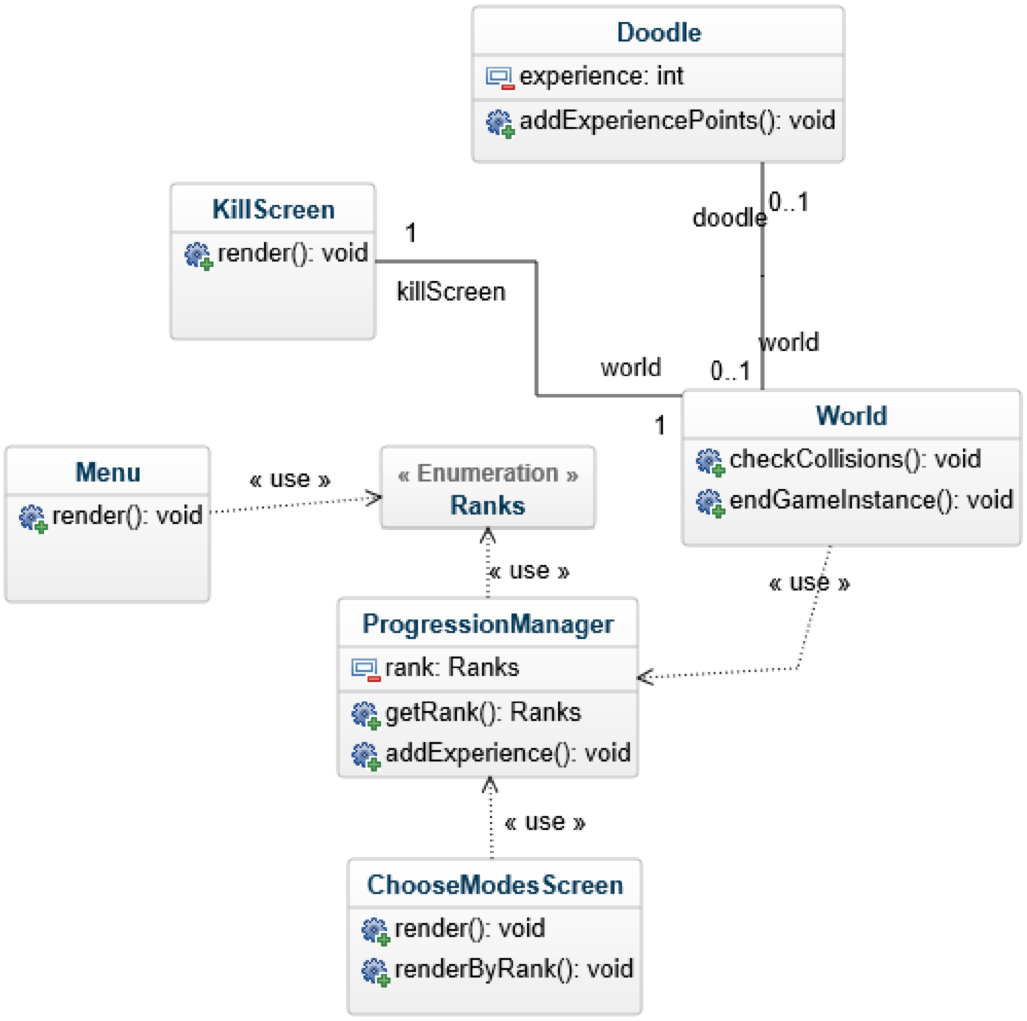
KillScreen

Purposes	Collaborators
Show the experience won in the before going game	Menu
Show the score won in the before going game	World

ChooseModesScreen

<u>Purposes</u>	<u>Collaborators</u>
Ability to choose the mode the player wants to play	ProgressionManager
Check if the rank is high enough to play the gamemode	

The UML for this implementation:


Doodle
experience: int
addExperiencePoints(): void
KillScreen
render(): void
1
killScreen
doodle
0..1
world
world
0..1
1
World
checkCollisions(): void
endGameInstance(): void
Menu
render(): void
« use »
« Enumeration »
Ranks
« use »
« use »
ProgressionManager
rank: Ranks
getRank(): Ranks
addExperience(): void
« use »
ChooseModesScreen
render(): void
renderByRank(): void