Assignment 5 - Group 8008 - SEM

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

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

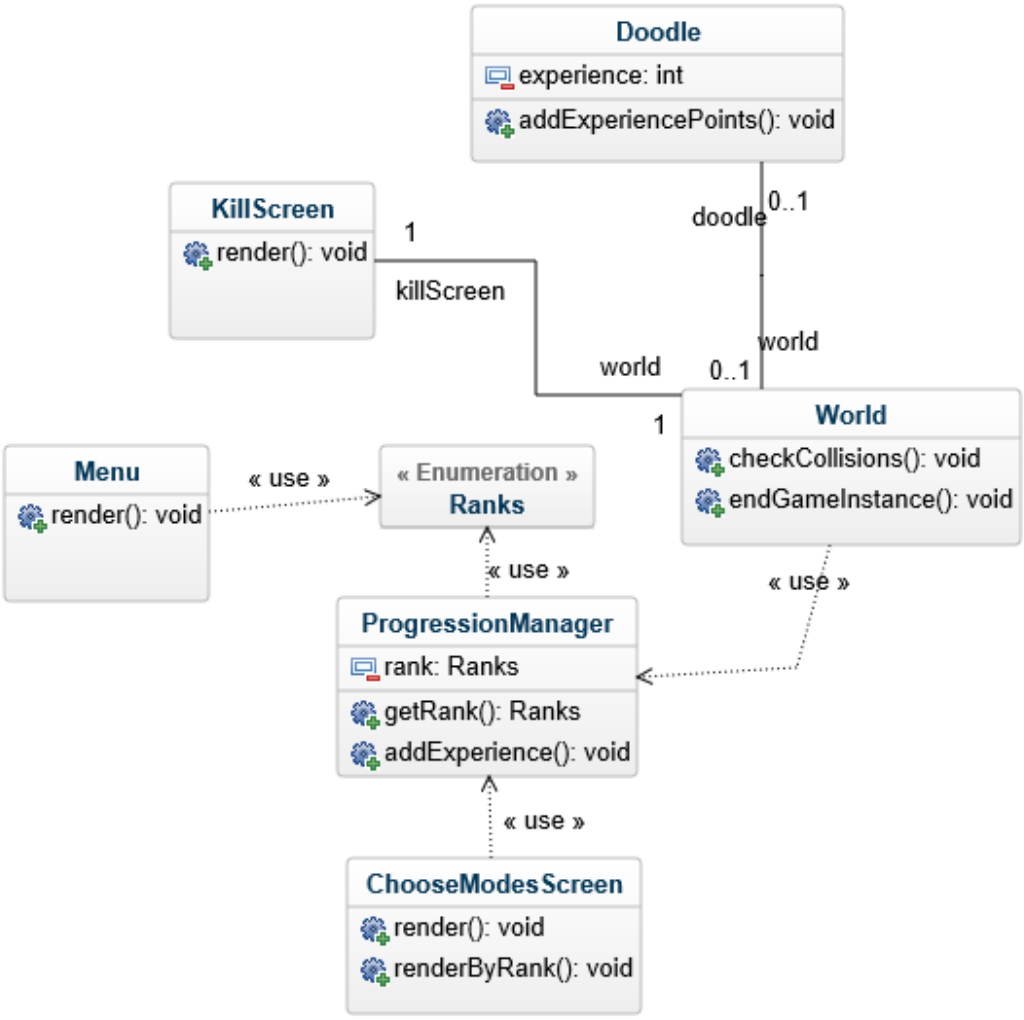
KillScreen

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

ChooseModesScreen

Purposes	Collaborators
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
0..1
world
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