Assignment 4

by

Group #6

Delft University of Technology

Faculty of Electrical Engineering, Mathematics and Computer Science.

Mekelweg 4,

Delft

Name	StudentId	Email
Clinton Cao	4349024	C.S.Cao@student.tudelft.nl
Michiel Doesburg	4343875	M.S.Doesburg@student.tudelft.nl
Matthijs Halvemaan	4353803	M.J.W.Halvemaan@student.tudelft.nl
Dmitry Malarev	4345274	D.R.Malarev@student.tudelft.nl
Sunwei Wang	4345697	S.Wang-11@student.tudelft.nl

TA:Valentine Mairet V.A.P.Mairet@student.tudelft.nl

Course: *Software Engineering Methods (TI2206)*

TU Delft

Table of Contents

A New Feature

For this assignment we received the task from our TA to implement a final boss (*Sharkelli)* for our game. Following RDD and using CRC cards we have derived the following cards:

End Boss	
Superclass(es): Entity	
Subclasses:	
Responsabilities	Collaborations
Represent the final boss of the game.	EntityFactory, Game

Weapon	
Superclass(es): Item	
Subclasses:	
Responsabilities	Collaborations
Represent the weapon that the player can use to fight the end boss.	PlayerFish, ItemFactory

The end boss is the final boss of our game; it's technically an entity, so therefore it has the responsibility of representing the final boss of the game. As the we have a factory for entities, this class will collaborate with the EntityFactory. Also it collaborates with the game, as our game will have a final boss.

The weapon is the weapon that the player will be using to fight the final boss; it's an item that the player can use. It therefore has the responsibility of representing the weapon that the player can use to fight the end boss. As the player is the only one that can use this item, it will collaborate with the PlayerFish class. Same as end boss, we have a factory for items, therefore it will collaborate with ItemFactory.

Comparing with our implementation

Comparing with our implementation, there is only one difference; we didn't name the weapon as *Weapon*, but as Lance. The reason is because we were brainstorming in our meeting on how to represent the fish of the player when the player has to fight the final boss. We came to the an agreement that it will be a "fish knight with a lance" and hence we named it "*Lance" instead.* We have also introduced the *Item* class back, as there are more items in the game than just fishbombs.

Using design patterns for the new classes

We decided to implement both *EndBoss* and *Lance* using the singleton design pattern. We chose for the singleton design pattern for both classes, because there should be at most one *Sharkelli* and on *Lance* item in the game.

Software metrics tools

First of all the analysis can be found in repository at Assignments -> Assignment 4.

Design flaw	Flawed method(s)	Why the error leading to the detected design flaw
Message Chains	handleCollisions()	The method uses one object to access another object, then uses the obtained object to access another object, and so on, all objects having different types. It impacts Complexity, Encapsulation and Coupling.
Data Class (Game)	setResX(int) setResY(int) setMusicPlayer(MusicPlayer) setStage(Stage)	The first two methods are unused during the entirety of the project. The last two methods were used, but were only used by the Game class itself, so they had no reason to be public methods. This caused the Game class to be treated as a data "container".
Feature Envy	makeAnimationTimer(...)	Nearly all data that was accessed by this method came from the MainScreenController. This led to less organized code.

We're happy to say all these errors have been fixed. However, in fixing the feature envy, another design flaw was introduced. The code from makeAnimationTimer (which collaborated too much with MainScreenController) was moved to the MainScreenController class. This fixed the feature envy, but in turn introduced a 'schizophrenic class' design flaw. Fortunately, the severity of this design flaw is only 1 (feature envy was a 4).

Of course we looked into fixing the schizophrenic class issue. The problem however stems from MainScreenController having too many responsibilities to begin with. In the last sprint, we spent much effort refactoring the code to improve the responsibility driven design. The animationTimer factory was actually an effort to take the responsibility of running the Game Loop away from MainScreenController. Because the AnimationTimer library is quite complicated, we did not manage to completely sever all the interconnectivity between MainScreenController and its influence in the AnimationTimer code (even when that code was placed in the AnimationTimerFactory), which is what lead to the feature envy design flaw.

What it comes down to in the end, is that MainScreenController currently contains all the logic and methods for the Game Loop which we want to have somewhere else. The way AnimationTimer works however has made it very difficult to accomplish this. Along with the fact that MainScreenController contains a ton of code (the refactoring of which would take a whole sprint at least, without even considering moving the AnimationTimer).

Issues regarding Maven packages

It has been pointed out by our TA that the packages in our repository does not follow the usual way on how it is supposed to be organised; there should a main package for all the classes, interfaces etc and another for the tests of our classes. This causes for the low coverage on Cobertura. I (Clinton) have done some research regarding the packages, but I couldn't find a solution. I tried creating new folders (as suggested on StackOverflow), but they ended turning into packages (but not the usual packages in Maven). I will still try to see if I can find a solution for this issue.