Assignment 4

Laura Pircalaboiu, Tim Anema, Ilya Grishkov,
Marc Otten, Ivar de Bruin

January 2020

Part I
Determining problems

1 Introduction

The tool we decided to use for our refactoring was the Metrics Reloaded plugin. We decided to evaluate the Chidamber-Kemerer metrics for this project, which include CBO (coupling between objects), DIT (depth of inheritance tree), WMC (Weighted Methods for Class), RFC (Response For Class), LCOM (Lack of Cohesion of Methods), NOC (Number of Children). Additionally, we considered Cyclomatic complexity, number of public methods, and number of parameters at the method level. We decided to consider these additional metrics since we felt like CK was not enough, and we wanted some orientation on what could be improved on on the method level.

2 Class-level metrics

2.1 Chidamber-Kemerer metrics

2.1.1 Coupling Between Objects (CBO)

For this metric we think the classes with over 15 classes coupled to them should be refactored. We settled on this metric by looking at the comparative number of classes in the entire project. Therefore, for the Ball, Cue, Table, Constants, Vector3f and SubmitScoreEndpoint on the server-side coupling is too high.

For the Constants class however, high coupling does make sense, the same goes for Vector3f as these are commonly used utility classes. Table is the center class for our 3d world and is therefore obviously also very coupled.
The ball and cue are used a lot during the game logic to put them in correct positions and therefore can't be improved even though they score high.
Lastly SumbitScoreEndpoint scores high but is impossible to improve.

2.1.2 Depth of Inheritance Tree (DIT)

For this metric, we believe we should only refactor methods where the DIT is more or equal to 3, such as CollisionHandler, Ball, HelpLine, MotionState and Plane. There are not a lot of problematic classes according to this metric, especially since all of them are library extension classes and that is not our job to refactor.

2.1.3 Weighted Methods for Class (WMC)

According to this metric, a very problematic class is again GameScreen. EightBall also has a high score, but that is because it contains all of the logic for 8-ball which is a complicated game.

2.1.4 Response For Class (RFC)

The Response for Class metric flags Gamescreen, EightBallTest, CueTest, GameTest, Bootstrap, WallFactoryTest, TableTest, SqlUserStore and NineBallTest and Game.

2.1.5 Number of Children (NOC)

This metric indicates a maximum number of children of 6, in EndpointTest. However, we don't think any of the classes need to be refactored with this method in mind since the only high NOC metric result belongs to a Test class, and is used to make testing easier.

2.1.6 Lack of Cohesion of Methods (LCOM)

This metric flags most of the classes in Screen. However, we believe that the methods in Screens belong together to a certain extent so we do not need to refactor with that in mind. The only exception might be, again, GameScreen.

2.2 Number of Public Methods

EightBall and NineBall both have over 10 public methods, and Ball has 6, Cue has 12 and GameScreen has over 25. WallFactory also has 9 public methods, all of these should be changed to protected or private to declutter the interface.

3 Method Level Metrics

3.1 Cyclomatic Complexity

The only two methods that were flagged by CC analysis were EightBall.checkRules() and GameScreen.CollisionHandler.onContactAdded() with a complexity of 11 and 13 respectively.

3.2 Number of Parameters

The complexity in Number of Parameters seems high in GameScreen.CollisionHandler.onContactAdded(), the constructor for the Table class and ScreenWriter, as well as the EightBall and NineBall constructors.

Part II
Solving the problems

1 Refactoring classes

1.1 Refactoring GameScreen

The GameScreen class had a coupling between object classes 52. The main issue was the fact that this class was taking care of a both rendering the game itself and and also making a lot of physics related calculations. We split the functionality of this class by extracting 3 new classes out of it. The original one is now responsible for all the rendering operations, and 3 additional classes, PhysicsEngine, InoutHandler, and CollisionHandler, take care of all physics calculations, handling keyboard and mouse input and detecting and handling collisions between game objects respectively. Those improvements allowed us to reduce CBO of GameScreen from 52 to 36. Those changes also fixed other code smells. Namely, by extracting classes we reduced the amount of public methods in the original one from over 25 to 16. It was mostly due to the fact that all methods related to handling input were moved to InputHandler class. Response for a class (RFC) was also reduced from 85 to 61, weighted method for class (WMC) from 51 to 31.

1.2 Refactoring to clean up interface

1.2.1 Game classes

The game classes contained 3 methods that where used by the general processGame() method and those should only be accessible to children not to all classes.
This improved the following things:

- Game: From 9 to 6
- EightBall: From 6 to 2
- NineBall: From 5 to 2

1.2.2 Gameobjects

The factories exposed methods other then the construct methods which should not be the case. These have all been changed to protected so they can still be used by either tests or other factories in the same package.
This means we had the following improvements in number of public methods.

- Ballfactory: From 7 to a constructor and a construct method.
- PlaneFactory: From 5 to 2.

- PocketFactory: From 6 to 2.
- WallFactory: From 9 to 5.

2 Refactoring methods

2.1 Refactor table constructor

We previously decided which balls to create in the Table constructor. Using extract method we split that into a separate method, which we then moved to the ballFactory using moveMethod. This last because it was feature envy and because switch statements should only occur in factories.

This refactor improved the CC of Table, reduced coupling and feature envy and got rid of a switch statement in a non-factory.

2.2 Refactor parameter lists

Certain classes had long parameter lists. We implemented builders for each of this classes to reduce their complexity.
Following methods were affected by those changes.

- Constructor for Table class
- Constructor for ScreenWriter class
- Constructor for abstract Game class
- Constructor for EightBall class
- Constructor for NineBall class

Even though CollisionHandler.onContactAdded() had 6 parameters it was impossible to refactor it as it overrides a methods in libGDX interface.

2.3 Reduce cyclomatic complexity

A number fo classes in the project had cyclomatic complexity of more than 10. One of the methods was responsible for checking rules of the game. It's complexity was reduced by extracting a method that would be responsible for only checking a rules related that if broken are leading to foul. The second method was responsible for detecting which object collided with each other by comparing their userValues. Certain conditional statements were moved to other methods to make the method more readable and reduce it's overall complexity. Following methods were affected by those changes.

- EightBallGame.checkRules(): From 11 to 6
- CollisionHandler.onContactAdded(): From 13 to 5