Code smells/Refactoring Assignment

We decided to combine exercise 2.1 and 2.2 into one because we think it's more logical this way. We used CodeMR and MetricsReloaded to compute the metrics. The thresholds we used to identify the code smells were not hard thresholds but a more subjective threshold. For example if we thought a method was too long we decided to refactor and extract it, not when the tools said a method was longer than x lines.

Method-level code smells

Change number	Code smell	Class/method in which it appears	Code metric before	Code metric after	Refactoring
1	Long method	GameScreenController constructor	CC: 15	CC: 1	Extract the complex code blocks into two separate methods.
2	High coupling	GameScreenController.onUpdate()	CBO:11	CBO:9	Moved asteroid splitting logic into Asteroid itself.
3	For testers only (van Deursen, et al. *Refactoring Test Code*)	Player	LOC: 239	LOC: 225	Change tests to not use the redundant methods and delete those methods.

4	Multiple return points[1]	GameScreenController.addBullet()
		
		Asteroid.spawnAsteroid()
			# of return points: 2
			
			3
				# of return points: 1
				
				1
					Refactor code to not need multiple return statements.

5	Long parameter list	Database.insertGame()	#of parameters: 5	#of parameters: 1	The method now takes the entire object as a parameter instead of the fields.

[1] Bugayenko, Y. (2015, August 18). Why Many Return Statements Are a Bad Idea in OOP. Retrieved from https://www.yegor256.com/2015/08/18/multiple-return-statements-in-oop.html?utm_content=buffer8474f&utm_medium=social&utm_source=twitter.com&utm_campaign=buffer

1. The GameScreenController constructor handles adding keyListeners to all the keys. The keylistener is a huge block of if statements that check which keys are being pressed. To make the constructor a lot more readable we extracted two methods from it. The first is the keyListener that is activated on key pressed (*GameScreenController.checkKeyPressed*), the second is the keyListener that is activated on key release (*GameScreenController.checkKeyReleased*). We moved these to two separate methods that each get called in the constructor. This makes the constructor a lot more readable and less complex.

2. Coupling Between Objects(CBO) reduced by removing asteroid splitting logic out of GameScreenController and into Asteroid. Asteroid returns an ArrayList of asteroids to create to the GSC.

Old code:

```
private void checkBullet(Bullet bullet
, Asteroid asteroid
, ArrayList<Medium> newMeds
, ArrayList<Small> newSmalls) {
  . . .
       if (asteroid instanceof Large) {
           Medium md1 = new Medium(new Random());
           Medium md2 = new Medium(new Random());
           md1.setLocation(asteroid.getLocation());
           md2.setLocation(asteroid.getLocation());
           newMeds.add(md1);
           newMeds.add(md2);
       } else if (asteroid instanceof Medium) {
           Small sm1 = new Small(new Random());
           Small sm2 = new Small(new Random());
           sm1.setLocation(asteroid.getLocation());
           sm2.setLocation(asteroid.getLocation());
           newSmalls.add(sm1);
           newSmalls.add(sm2);
       }
  . . .
}
```

New code:

```
private void checkBullet(Bullet bullet
, Asteroid asteroid
, ArrayList<Asteroid> newSmalls) {
  . . .
  . . .

   newAsteroids.addAll(asteroid.split());
  . . .
  . . .
}
```

By refactoring the code this way the cyclomatic complexity of the method goes down and it will be a lot easier to add more different types of Asteroids in the future. Before you had to add a new if statement to the method. Now you can just implement the split method in the asteroid subclass.

3. Some classes contain methods that are only used in tests, these methods are not necessary for the program to function and only add extra complexity and size to the codebase. By removing them the Lines Of Code (LOC) decreases and the code becomes less complex. Examples are Player.setLives() and player.setTotalScore.

4. The GameScreenController.addBullet() class contained 2 return points because it did a null check, when this returned true the method would immediately return. To remove the need for 2 return statements we moved the functional code into the null check and negated the conditional in the if statement.
   Asteroid.spawnAsteroid had 3 return points (1 for each asteroid type). To remove the need for these return points we introduced a local variable and returned at the end of the method.

5. The insertGame() method had as parameters all the fields of the Game class, so now we have changed it to take directly the Game object as parameter.

Class-level code smells

Change number	**Code smell**	**Class/method in which it appears**	**Code metric before**	**Code metric after**	**Refactoring**
1	Inappropriate intimacy	GameScreenController/AudioController	WMC: 106	WMC: 96	Move method
2	Dead code	Test.java.integration: -.database.DatabaseUserTest & -.authenticationservice.AuthenticationServiceIntegrationTest	Metrics are for the whole test suite LOC: 1745	LOC: 1612	Delete the dead code
3	Speculative generality	Java.models.Asteroid Java.models.SpaceEntity	LOC: 158 LOC: 170	LOC: 157 LOC: 162	Delete the unused field Delete the unused method
4	Switch statements	Java.models.Asteroids	LCOM: 1.063	LCOM:0.946	Extract method
5	Middleman	Java.Database.insertGame()	MPC: 123	MPC: 122	Extract method and delete delegating method
6	Large class	Java.controllers.GameScreenController	LOC: 315 WMC: 114 RFC: 159 NOM: 31	LOC: 268 WMC: 103 RFC: 164 NOM: 25	Extract the methods dealing with the graphic part in another class.

1. Inappropriate intimacy of GameScreenController with AudioController. This was resolved by moving the logic out of GameScreenController to new methods in AudioController. This lowered Weighted Method Count (WMC), the amount of method calls, to 96.

2. We have some integration tests that were present since the early stages of development, but these were commented out because they couldn't run on the CI/CD pipeline. This made our test suite more complex than necessary so we deleted them. After deletion the LOC of the test suite went down from 1745 to 1612.

3. In the Asteroid class the largeSpawnTreshold field is declared but it is never used. Throughout our development process we realized that it is not needed, but we never removed it. Now, by finally removing it, our code is clearer and there are no unused fields, values that can confuse us.
   In the SpaceEntity class the setImage setter was generated while generating all the setters and getters, but was never used, so we removed it.

4. Moved the switch statement form the constructor of the Asteroid class to a separate method dedicated to deal with it and return into the constructor the final decision. This way our code if better organized and the constructor doesn't have to deal with making decisions, it can focus on its main responsibility.

5. The insertGame() method that took the Game object as parameter simply called the insertGame() method that took as parameters the fields of the same object. This way we had one method that wasn't actually doing anything other than delegating its task to another method. We fixed this by keeping the method that took the Game object as parameter (keeping method level code smell #5 in mind) and extracting its fields inside the method and then applying the logic from the previous method, finally removing the delegating method. We have used the MessagePassingCoupling(MPC) metric.

6. The GameScreenController was a very large and complex class that was taking care of both logic and design of the Game screen. For this reason we decided to create a GameScreenView class in the *views* package where we stored the methods dealing with the creation of graphic content on the screen, such as: *createContent()* or *addSpaceEntity()*. By doing this the complexity of our went down and the code results to be cleaner than it was before.

Code smells theory

Code smell	Code metric used	Refactoring
Long method	Cyclomatic complexity(CC), lines of code	Extract method
Large class	CC, LackOfCohesionOfMethods, Connectivity metric	Extract class
Long parameter list	Nr. of parameters	Introduce parameter objects
Feature envy	Coupling between objects(CBO), Message passing coupling	Move method refactoring
Coupled classes	CBO, DepthOfInheritanceTree, NrOfChildren	Move method, Extract class, Replace delegation with inheritance
Switch statements	CC	Replace conditional with polymorphism
Multiple return points	RETURN	Refactor method to use a single return point.

Metrics Before and After refactoring

Class metrics	CBO Coupling between object	LCOM Lack of Cohesion of Methods	NOC Number of Children	RFC Response for Class	WMC Weighted Method Count	CSO Class size (operations)	DIT Depth of Inheritance tree
Average before refactoring	6.54	2.96	0.38	27.25	15.78	32.78	1.62
Average after	5.32	3.00	0.40	24.85	14.44	31.00	1.64

Method metrics	iv(G) Design complexity	LOC Lines of Code	NBD Nested Block Depth	v(G) Cyclomatic Complexity
Average before refactoring	1.62	10.79	0.33	1.78
Average after	1.60	10.85	0.35	1.72