TI1705 - Part II

Gerlof Fokkema - 4257286

May 26, 2014

4 Part II

4.1 Mocks

Exercise 9 *Write a test suite for the level.MapParser class. Start out with the nice weather behavior, in which the board contains expected characters. Use mockito to stub the factories, and use mockito to verify that reading a map leads to the proper interactions with those factories.*

Please refer to *nl.tudelft.jpacman.level.MapParserTest* for the implementation of the JUnit test.
The factories that had to be stubbed were the *LevelFactory* and the *BoardFactory*. Verifying proper behaviour was done by giving the *MapParser* a predefined map, for which we know the amount of squares. After that we can count how often the *MapParser* has called specific functions and we can match that against the known amounts.

Exercise 10 *Extend the test suite to bad weather situations, which should raise the proper exceptions.*

While extending the test suite to bad weather situations, we encountered a problem with the way EclEmma calculates code coverage. EclEmma is not able to calculate code coverage for lines that throw exceptions, therefore the reported code coverage score is significantly lower than the real score.

4.2 Testing Collisions

Exercise 11 *Analyze requirements (doc/scenarios.md) and derive a decision table for the JPacman collisions from it.*

After analyzing the requirements in *scenarios.md* and the implementation in *PlayerCollisions* we concluded that Units are subdivided in 2 types for collision maps: the *mover* and the *movedInto* unit.
In JPacman, there's 2 types of Units that can be a *mover*:

- a *Player*
- a *Ghost*

A *mover* can then collide with Units of all types:

- a *Player*
- a *Ghost*
- a *Pellet*

Putting all this together we get the following decision table:

			Rules					
Conditions	Mover	Player	Y	Y	Y	-	-	-
		Ghost	-	-	-	Y	Y	Y
	CollidedOn	Player	Y	-	-	Y	-	-
		Ghost	-	Y	-	-	Y	-
		Pellet	-	-	Y	-	-	Y
Actions	Player dies			X		X		
	Pellet is taken				X			
	Player gets points				X			

Exercise 12 *Based on the decision table for collisions, derive a JUnit test suite for the level.PlayerCollisions class.*

Please refer to *nl.tudelft.jpacman.level.PlayerCollisionsTest* for the implementation of the JUnit test.

Exercise 13 *Restructure your test suite from exercise 12 so that you can execute the same test suite on both PlayerCollision and DefaultPlayerInteractionMap objects.*

To be able to execute the same test on multiple *CollisionMaps*, we've chosen to implement *PlayerCollisionsTest* as a parameterized JUnit test. This means we won't be able to use the *MockitoJUnitRunner*. Therefore we initialize Mockito manually instead.

Exercise 14 *Analyze the increase in coverage compared to the original tests in jpacman-framework, and discuss what collision functionality you have covered additionally, and which (if any) collision functionality is still unchecked.*

The coverage percentage we reported in the previous lab exercise (excluding test cases) was 79%. After implementing the test cases for this lab exercise, the overall coverage percentage has risen to 86%. An overview of what has changed: Additionally covered functionality mainly includes the alternative collision han-

	Original coverage %	New coverage %
PlayerCollisions	100 %	100 %
CollisionInteractionMap	0 %	97 %
DefaultPlayerInteractionMap	0 %	100 %
Overall	81 %	87 %

dler, the *DefaultPlayerInteractionMap*. Of this collisions handler, we've now tested most functionality using multiple colliders and collidees.
Although the coverage percentages are now quite high, there is still untested functionality. An example is what happens when a *Player* moves onto a square that contains both a *Ghost* and a *Pellet*. Other examples include the functionality of symetric collision handlers and unexpected situations like the *CollisionInteractionMap* not having a registered collisionHandler.

4.3 Submit Part II

Exercise 15 *Submit Part II as you submitted as described in Section 3.3.*