TI1705 - Part I

Gerlof Fokkema - 4257286
Joris Lamers - <studentnr>

May 2, 2014

3 Part I

Your introduction goes here!

3.1 End-to-End Testing

Exercise 1 *Execute the smoke test, with coverage enabled. What overall (line) coverage percentage do you get? Name 2 classes that are not well-tested, and explain why the smoke test does not cover it.*

The overall coverage percentage for the smoke test is 79% (test classes excluded). One thing to note is that the line coverage varies slightly between runs. This is because testing is done using the interactive GUI. Therefore the actions that are tested are slightly influenced by the random movements of the pacman ghosts. Two classes that are not well-tested are:

- nl.tudelft.jpacman.level.CollisionInteractionMap
- nl.tudelft.jpacman.level.DefaultPlayerInteractionMap

The reason for the limited coverage in those classes is that the GUI is only started for a limited time. These two classes are both involved in collision handling. Because JPacman is only started for a limited time no collision scenario's are encountered during the tests.

Exercise 2 *Study the acceptance scenarios for User Stories 1, 2, and 4. Turn each of them into a test case, as far as possible. To do so, use the approach adopted in LauncherSmokeTest to start the game and trigger specific behavior.*

No specific remarks.

Exercise 3 *Which functionality of story 2 was hard (or even impossible) to test? Why?*

Scenario "S2.5: Player wins" was hard to test.
We can't simply automate the test scenario for S2.5, because it would involve writing a whole algorithm for a non human player controlling JPacman.
One way we can test it however, is by looping our JUnit tests until a human player eats the last Pellet. We can then see whether the JPacman framework will correctly decide the game has been won. This is how we've implemented the test for Scenario S2.5.

Exercise 4 *Now try the same for Story 3 (moving monsters). Why are these scenarios hard to test?*

3.2 Boundary Testing

Exercise 5 *Provide a domain matrix for the desired behavior of the boundary values in the withinBorders method.*

"x >= 0 && x <getWidth() && y >= 0 && y <getHeight()"										
Boundary			Test Cases							
Variable	Condition	Type	t1	t2	t3	t4	t5	t6	t7	t8
x	>= 0	on	0							
		off		-1						
	<getWidth()	on			val					
		off				val				
	typical	in					1	2	3	4
y	>= 0	on					0			
		off						-1		
	<getHeight()	on							val	
		off								val
	typical	in	1	2	3	4				
Result										

Exercise 6 *Implement the corresponding test by using JUnit's Parameters annotation.*

3.3 Submit Part I

Exercise 7 *In your report, analyze whether the code is ready for submission: Explain check-style violations that remain (if any), provide a log of all tests passing, and include a brief assessment of the additional adequacy achieved in the jpacman-framework thanks to your new classes. Also reflect on your continuous integration server results, and your commit behavior.*

Exercise 8 *Create a release with appropriate version number in your pom file, git tag and push. Submit the zip and report to CPM as well.*