UNIVERSITY OF ANTWERP
DEPARTMENT OF MATHEMATICS AND COMPUTER SCIENCE

Software Testing Lab

Assignment 4

Submission Deadline: **March 20th, 20:00**

Refer to Assignment 1 for introductory information.
Note that this assignment has a 25 point late fee instead of additional exercises.

DECISION STRUCTURES

In this exercise we will test the movement of guests using decision structures (Forgács Chapter 7, p129-154), and extend JPacman with moving monsters.

- **Exercise 1.** Create a decision table following the style of Table 7.6 (Forgács) indicating what should happen when a guest tries to occupy a new cell. Cases to be distinguished include whether or not the move remains within the borders, whether or not the move is possible based on the type of the moved object (player or monster), and the type of the (optional) guest occupying the other cell. **(Required, 10 points)**

- **Exercise 2.** Run the current test suite and describe the coverage of `Move`, `PlayerMove`, `Guest`, and all `Guest` subclasses. **(Required, 10 points)**

- **Exercise 3.** Implement all entries in the decision table concerning player movements as JUnit test cases in `PlayerMoveTest` class. Since the player movement has been implemented already, start by testing these. **(Required, 10 points)**

Table 7.6 Extended-entry decision table for the TVM example

Conditions	**R1**	**R2**	**R3**	**R4**	**R5**	**R6**	**R7**	**R8**	**R9**	**R10**	**R11**	**R12**	**R13**
Number of sel. tickets, No	1 ≤ No ≤ 10								No = 0				10 < No or No < 0
Standard ticket	Y	N	N	Y	Y	N	Y	N	Y	–	–	N	–
Short distance ticket	N	Y	N	Y	N	Y	Y	N	–	Y	–	N	–
24-hour ticket	N	N	Y	N	Y	Y	Y	N	–	–	Y	N	–
Actions													
Payment possible	X	X	X										
Total price (EUR)	No x 2.1	No x 1.4	No x 7.6										
Any ticket type is selectable												X	
Ticket selection error, logging				X	X	X	X	X	X	X	X		X

- **Exercise 4.** Re-run with coverage enabled, and re-assess the coverage. **(Required, 10 points)**

- **Exercise 5.** Explain the interplay between the abstract methods `Guest.meetPlayer` and `Move.tryMoveToGuest` and their implementations in `Guest` and `Move` subclasses. **(Required, 10 points)**

- **Exercise 6.** Implement a monster move in the same style as a player move. Add a `MonsterMove` class, place it correctly in the inheritance hierarchy, and implement the required methods. Make sure you add or update appropriate invariants as well as pre- and post-conditions wherever possible, and implement them using assertions. **(Required, 20 points)**

- **Exercise 7.** Introduce a `MonsterMoveTest` class to implement the test cases related to monster moves. You will probably want to extend `MoveTest` for this. Verify the test coverage for this class. **(Required, 20 points)**

- **Exercise 8.** How many tests in your decision table would you need to get 100% coverage of the relevant moving methods? Why do you need the remaining test cases? **(Required, 10 points)**

Late fee: -25 points