SWEA Iteration 9: Distribution using Broker I

`<<Group name>>`
Computer Science, University of Aarhus
8200 Århus N, Denmark
<<Names>>

<<Date>>

1 Broker 1.1: TDD of pass-by-value Game methods

[Shortly outline your group's process with developing the missing broker roles: which test stubs did you use and for what purpose, use of visual clues (System.out) during the process, etc.?]

1.1 ClientProxy Implementation

[Insert one/two examples of your ClientProxy implementation of Game methods]

[Short explanation of the code]

1.2 Invoker Implementation

[Insert fragments of your Invoker code, ideally matching those selected above]

[Short explanation of the code]

1.3 Testing

[Insert a few of the most important JUnit test code]

[Short explanation of the test code, and the doubles used]

2 Broker 1.2: TDD of Card and Hero methods

[Shortly outline your group's process with developing the missing broker roles: which test stubs did you use and for what purpose?]

2.1 ClientProxy Implementations

[Insert one/two examples of your ClientProxy implementation of Card and Hero methods]

[Short explanation of the code]

2.2 Invoker Implementation

[Insert fragments of your Invoker code, ideally matching those selected above]

[Short explanation of the code]

2.3 Testing

[Insert a few of the most important JUnit test code]

[Short explanation of the test code, and the doubles used]

3 Broker 1.3

3.1 Screenshots and Explanation of Story Test

[Create and include screenshots of the following user story:]

First, we start the server on a computer with IP [what is the ip, if you are not using 'localhost'?].

[include image of shell on server computer in which IP is clearly visible; and next the start command of the server]

Next, we start the 'hotstoneStorytest' on another computer/another shell and connect it to the server

[include image of shell on client computer which connects to the server, executes the story, and print the relevant method return values]

After the client has executed the story, the server has printed relevant received requests and returned replies as log messages:

[include image of log output on the server computer]

4 EC test of SigmaStone augmentMinion()

4.1 Conditions

Regarding the $augmentMinion()$ function

$$(attack, health) = augmentMinion(minion, field, hero)$$

the following conditions are relevant:

- minion class
- field support
- *(Your group's further analysis here)*

4.2 EC tables

These leads to a first version of the *equivalence class table*:

Condition	Invalid ECs	Valid ECs
minion class (set type)	-	Medi [a1]; Nord [a2]; Asia [a3]; Cent [a4]
...	...	...

(You may provide elaborations of the table in the likely event that some ECs require further partitioning. Hint: As the function includes aritmetic computation, be sure to read the FRS computation heuristics.)

The found ECs have the *representation* and *coverage* properties because [argument here].

4.3 Test case table

We use the heuristics generate test cases as outlined in the extended test case table:

ECs covered	Test case	Expected output
...	...	...

5 Backlog

[Outline backlog items if any]