The following testplan**s** are to be run and their logs submitted to cuLearn (among other stuff).
You should add to the players' input whatever your networking requires: we're only giving the gist of what has to be tested. For example, your game should start by asking for the number of players (from 2 to 4) and the number of rounds.
These test plans are NOT exhaustive: see the marking scheme to see what else you must address!
Finally, we do not expect you to test for robustness (eg attack + defense times more than 4, inexistent player, etc.) EXCEPT for too many (i.e., 5) players.

//**test plan 1** (with networking ONLY if it works)
//start your game with 2 players and 7 rounds
//have joe and fred join
//round 1
select joe fred thrust 1 dodge 3
select fred joe thrust 1 duck 3
roll fred 1 //real attack direction is thrust – hits on speed
roll joe 1 //real attack direction is thrust – hits on speed
//display outcome and indicate round 2 starts
select fred joe thrust 2 swing 2
select joe fred thrust 3 duck 1
roll joe 2 //real attack direction is thrust – no hit
roll fred 2 //real attack direction is thrust – no hit
//round 3
select fred joe thrust 3 charge 1
select joe fred thrust 3 charge 1
roll joe 2 //real attack direction is thrust – hits on direction
roll fred 1 //real attack direction is thrust – hits on direction
//round 4
select fred joe thrust 3 duck 1
select joe fred thrust 3 charge 1
roll joe 3 //real attack direction is smash – hits on direction
roll fred 6 //real attack direction is swing – hits on direction
//round 5
select fred joe swing 3 duck 1
select joe fred smash 3 dodge 1
roll fred 1 //real attack direction is swing – hits on direction
roll joe 1 //real attack direction is smash – hits on direction
//round 6
select fred joe swing 3 charge 1
select joe fred smash 3 charge 1
roll fred 4 //real attack direction is thrust – hits on direction
roll joe 5 //real attack direction is thrust – hits on direction
//round 7
select fred joe thrust 1 dodge 3
select joe fred smash 1 charge 3
roll fred 1 //real attack direction is thrust – hits on direction and speed
roll joe 4 //real attack direction is swing – hits on direction and speed

test plan 2 (3 players)
//start your game with 3 players and 3 rounds
//have jack, joe and fred join
select jack joe swing 1 duck 3
select fred joe thrust 2 dodge 2
select joe fred smash 2 charge 2
roll fred 1 //real attack direction is thrust – hits on direction
roll joe 4 //real attack direction is swing – hits on direction
roll jack 1 //real attack direction is swing – hits on speed
//round 2
select joe jack smash 3 duck 1
select jack fred thrust 3 charge 1
select fred joe swing 3 dodge 1
roll fred 1 //real attack direction is swing – no hit
roll joe 1 //real attack direction is smash – no hit
roll jack 1 //real attack direction is thrust – no hit
//round 3
select fred joe thrust 1 dodge 3
select joe jack swing 2 duck 2
select jack joe smash 3 charge 1
roll fred 6 //real attack direction is swing –hit on speed
roll joe 4 //real attack direction is thrust – hit on direction
roll jack 3 //real attack direction is swing – no hit

test plan 3 (4 players)
//start your game with 4 players and 1 round
//have jack, joe , jill and fred join
select jack joe swing 1 duck 3
select fred joe thrust 2 dodge 2
select jill fred smash 3 charge 1
select joe jill smash 1 charge 3
roll fred 1 // hit on joe
roll jack 4 // hit on joe
roll jill 5 // no hit
roll joe 6 //hit on jill
The 3 testplans supplied above, I repeat, are only meant to quickly assess what you got working. Most of your marks are earned from evaluating your ability to respect a TDD approach. Recall I have specified the order in which I want tests and then corresponding code to be submitted to BitBucket. Given this order, I explain below how your assignment will be marked:

TENTATIVE MARKING SCHEME:

1) GAME LOGIC 50 marks (BTW attack *time* and attack *speed* mean the same thing)
Out of these 50 marks, a successful testplan 1 is worth 2 marks, a successful testplan 2 is worth 4 marks, as is a successful testplan 3, for a total of 10 marks. Any testplan with errors or not run is worth 0.

The game logic, what Howard calls the rule engine, needs to be developed first and completely independently of the client/server.

Looking at your repository, and following the posted order in which I said features had to be developed, we want to see JUnit tests for (0 to 5 marks per bullet):

- your game engine concluding an attack hits because of time: you should check at least one such case. More systematically, regardless of attack and defense directions, you should try all possible combinations of attack time and defense time (viz 1-3, 2-2, 3-1)
- your game engine correctly figuring out the actual direction of an attack: there are 9 tests required here (3 categories of rolls against each of the 3 attacks).
- your game engine correctly figuring out if an actual attack direction matches or not a defense: there are again 9 tests required (three defenses to try for each attack).
- your game engine concluding an attack hits when both time and direction cause it to hit. One test is enough but be more thorough is time permits.
- your game engine concluding an attack does not hit. Again, one example will be enough but do write more tests time-permitting.
- your game engine correctly resolving the attack of a second player and reporting the number of hits inflicted to each of these 2 players: try no hits, 1 hit to first player, 1 hit to second player, 2 hits.
- your game engine correctly resolving attacks for 3 players:
    - try all different targets: with no hits, some hits, all hits
    - try two players with the same target, with no hits, 1 hit, 2 hits
- your game engine correctly resolving attacks for 4 players:
    - try all different targets: with no hits, some hits, all hits
    - try two players on one target, and the other players targeting one of the first 2
    - try three players with the same target, with no hits, 1 hit, 2 hits, 3 hits

Your handling of several rounds is tested via the testplans, as is different order of submissions of selections and rolls.

We DO expect you to have all tests you wrote pass. We will run your game to verify this.

2) CLIENT and SERVER 50 marks

Out of these 50 marks, testplan 1 run successfully **with networking** is worth 7 marks, testplan 2 run successfully with networking is worth 7 marks, and testplan 3 run successfully with networking is worth 6 marks, for a total of 20 marks. These testplans test the integration of the game engine with networking.

You must show the Server and Client being developed using TDD
There are two ways to do this:
OPTION 1

Use a Mock Server that simulates managing connections and implementing the rules

OR

OPTION 2 (demoed by Howard – see posted code)

1: You create a client test (e.g. connect(iptaddres, port) and a corresponding empty method in the client. This will fail since there is no server code yet

2: You then write just enough code in the server for it to start up and accept a connection - including the logging requirements. Then you write the corresponding code in the Client class to connect.
Repeat and test for multiple clients

3: Write a client test case for selecting an attack/defense
Write the corresponding code in the server - including the logging requirements
Write a client test case for rolling
Write the corresponding code in the server - including the logging requirements
Write the server code for reporting outcomes at the end of a round (and either quitting the game on the last round or starting the next round)

Here the ipaddress and port of the server are the ones that the clients "initiate" their connection with: the server does not take part in the game.

We DO understand there is a multitude of variations possible for networking, so it's what you test and in what order that matters, not the how per se.

Regardless:

- **you should echo any player input to all others**
- **you should broadcast the number of players and rounds at the start of the game**
- **you should broadcast the start of each round and the end of the game**

For option 2, here's the suggested approach for testing your networking using TDD:
(again: if you do implement networking differently, try to duplicate WHAT is being tested in what order)

We start with three "empty" classes

1: Server Class
2: Client Class
3: Client Test Class (JUnit)

Step 1: Test one Client connection
--- This test is meant establish a connection with the server
--- And it will fail at first

TEST 01

ClientTest
@Test
connect (ipaddress, port)

Client Class
method Boolean connect (String IPAddress, Integer port);

Server
void startup (Integer Port)

Now: the test will fail since there is no code

Write the client code to connect first (and it should fail)
Then write the actual server code to accept a connection AND THE APPROPRIATE Log
--- Server : ipAddress:port started (date and time required)
--- Client ipaddress:port attempted to connect (date and time required)
--- Client ipaddress:port connected (date and time required)
--- SENT Message to Client [ipaddres:port]: Logged in on (datetime)

--- if you notice there is no separate thread spawned for the client
--- The server should return a message to the client indicating
-> Successfully connect to server [ipaddress:port] and IP Address of client
The client connection test should now pass

END TEST 01 [5 MARKS]

TEST 02
For this test we just want to make sure we can create multiple client connections

ClientTest
@Test
testMultipleConnections (ipaddress, port)

This test should initialize 2 or more clients

All clients should be able to connect successfully:
--- However they cannot communicate with each other since they are using the same port
--- This should be covered in the test

END TEST 02 [5 Marks]

TEST 03
Now we want to test creating multiple connections and sending a message from the server
Note: We are not interested in testing the rules engine yet

Now write the client thread and required code in the client class and test it
--- It should fail
--- Now write the code in the server class that will spawn a server thread for each
--- client and send a message back to each client
--- Again this is tested for a single client and then multiple clients

Here we have three specific tests that must occur
1: Each client must connect
2: Each client must receive a message from the server

END TEST 03 [5 Marks]

TEST 04 [5 Marks]
Repeat with 3 players

TEST 05 [5 Marks]
Repeat with 4 players

TEST 06
// Now we want to test minimum and maximum number of clients
Client Test Case
Test establishing 2, 3, 4 and then, time permitting, try 5 connections (1 too many)

ClientTestCase

testMaxmimumConnections (Integer maxNumberOfConnections)

--- There are no changes to the client side code
Only the server side code

--- You must modify the server code to handle min and maximum number of connections
--- and return the appropriate message(s) to the client
--- i.e. SUCCESS MESSAGE (as above)
--- ERROR MESSAGE -> Maximum number of connections exceeded
--- ALL connections and messages to/from the client must be logged to the server log file

END TEST 06 [5 MARKS]

Then your repository should address the integration of networking with the game engine, which is what the testplans test.

Bit Bucket ids:
matt_preston
michaelhum
AndreiJuc

Your culearn submission must be a single zip file called <your LastName>.zip
This file must include:
- a readme.txt file that summarizes what works and what does not in your code, as well as explains your networking solution (including which versions of dependent code you use).
You must also explain in that file what are your test suites and how to run them. As demoed by Howard, you must have a StartServer class to make starting the system directly from Eclipse very simple.
- all your source code (including for tests, test suites, test runner).
- all jars including what you depend on (eg log4j, hamcrest, etc.) as there are different versions of the latter… The TA wil run the client code using a jar (i.e., not from Eclipse) via:

    java -jar clientapp.jar

Submit accordingly…
On this topic, Howard suggests this tutorial on how to create an executable jar file
http://java67.blogspot.ca/2014/04/how-to-make-executable-jar-file-in-Java-Eclipse.html
- the logs for each of the 3 testplans, names testplan1Log testplan2Log testplan3Log

With respect to test code, your TA must be able to run your test suite(s) directly in Eclipse, which implies the use of a test runner, as demoed by Howard. The TAs will NOT run individual tests: you must use test suites and explain what they are in your readme file.