Iteration 2 requirements

a) Documentation requirements: (20%) (do study example projects!)

- *Use cases* and their scenarios with **traceability** back to the rules (to demonstrate coverage) + a use case diagram that captures play between player**s**. Use the STD format of the slides for the use cases. (5 marks)
- One or more class diagrams + a CRC card for each class (2 marks)
- UML 2.0 interaction diagrams **corresponding** to your UC diagram and each use case: it is this correspondence that is essential. (5 marks)
- Explanation of networking strategy and implementation: what does a client do? What does the server do? How do they exchange information? Pros and cons of your networking strategy? (4 marks)
- Explanation of your design: (4 marks)
    a. Overall architecture
    b. Use of patterns: motivation for use
    c. How much did you refactor from iteration 1 and what kind of refactoring did you perform (elimination of duplicated code, redistribution of responsibilities, etc.)
    d. Explanation, pros and cons of your design

b) **Integrated networking is a must**: you **must** demonstrate one multiplayer multi-round game **over several machines**.
You must support between 2 and 5 players.
You get a mark between 0.5 and 1 for networking and that mark is multiplied with the one for game logic below. (That is, no networking reduces your mark for game logic by half!)

c) **An integrated GUI is a must**: It is essential to display the cards of each player in the order in which they were played. Also, SHIELD and STUNNED must be correctly displayed separately from other cards.
You get a mark between 0.5 and 1 for your interfacing and that mark is multiplied with the one for game logic below. If you use a textual interface, your mark for the interface will be 0.7 as a textual interface is less work than a GUI.

d) **Game logic (to be tested without networking) (55%)**
Marks are given by you listing which JUnit test(s) address(es) each of the following AND showing that these tests have passed.

- for each tournament, *minimally* a Junit test for each of the following: (20%)
    a. one player draws/starts, others draw but do not participate (ie withdraw)
    b. one player draws/starts, others draw but only one participates by playing i) a card and ii) several cards
    c. one player draws/starts, others draw and some participate by playing i) a card and ii) several cards
    d. one player draws/starts, others draw and all participate by playing i) a card and ii) several cards

e. starting with i) a supporter and ii) several supporters
f. a multiplayer tournament has several rounds where each player plays one and then several supporters in different rounds
g. trying to play cards that do not get the current player to beat the tournament originator (ie not en ough to be the leader)
h. restriction to 1 maiden per player per tournament
i. winning and getting token
j. winning and choosing token when purple tournament
k. losing with a maiden and losing a token

- for each action card, *minimally* a Junit test for: (25%)
  a. playing this card on an unshielded player
  b. playing this card on an shielded player
  c. undoing this card using Ivanhoe
  d. checking a used action card is indeed thrown away
- scenarios: *minimally* a Junit test for: (10%)
  a. the player who should start *cannot* start a tournament
  b. last tournament was purple, cannot be purple again
  c. trying to play an insufficient number of cards to become the leader on my turn
  d. trying to play invalid cards (wrong color)
  e. coming to end of deck
  f. using CHARGE in a green tournament with every player with only green 1s: one card must remain
  g. other example of overriding rule: at least one card must remain
  h. winning the game
  i. the deck uses 110 cards

d) AI Strategies: (10%)

You must explain in your design at least 2 distinct AI strategies.
There must be a set of JUnit tests to 'prove' the correctness of each of these strategies. Your design must be able to accommodate may more AND it must support one human player playing against one or more AI players.
You can come up with your own strategies OR use the following conceptualization for simple ones: Below, a strategy is taken as combining 3 distinct decisions:

- *what color to start with*: that of missing token, or of what I have most of or (card counting) the one with the highest likelihood of winning (whichever way this is computed..) or anything but what another player needs to win, etc.
- *what first card(s) to play*: try to win with minimum played per turn to maximized picking up cards, or try to win with a (hopefully) decisive first set of cards to play (ie blow the competition away)
- *what card(s) to play (except to start a tournament)*: none (ie you figure it's not worth it or you will likely lose so you withdraw) + 2 of previous bullet
- so there's at least 3 * 2 * 3 = 18 different combinations of such simple strategies

Useful Bells and whistles: (<= 20% i.e., how to get an A or A+)

- Nice GUI features such as: resizable cards, disabling cards that cannot be played, etc. (~5%)
- Robust networking: eg., handling loss of a player, etc. (~5%)
- Robustness of game: eg., preventing playing out of turn, etc. (~5%)
- Did you manage to write at least one test procedure (or set of test procedures) that automates playing a game **over the network**? (~5%)

Other bells and whistles will be considered BUT will be scored wrt their usefulness in playing the game… Maximum mark: **20%** (ie you can score 105% without the bonuses!)

If you are working alone, you are **not** expected to do the AI strategies or "Bells and whistles": game logic is worth 80% of your mark, but networking and GUI are marked as explained above.

Bonuses: (no maximum mark BUT do watch out for the learning curve…)

- jUCMNav bound UCMs **corresponding** to your use cases and use case diagram (<=5%)
- Abbott tests for your GUI (<=8%) - if not claimed in iteration 1)
- Cucumber tests (<= 10%)
- Scripted tests (ie automating the use of your GUI): you choose a tool… (<=10%)
- A general approach to automate the testing of games over the network (<=10%)
- Real-time Ivanhoe: no player order IN **a** tournament: one player starts a tournament but then players draw/play concurrently until one wins the tournament ☺ (<=15%) This may be either integrated as an option in your game OR offered as a second separate version of the game. The onus is on you to demo that it works correctly.

Bonuses should be attempted only once complete game functionality, networking and GUI have been satisfactorily completed.

Do remember that we DO expect a virtual image on a USB stick at the time of your demo on April 7th.

A correction grid to fill out, as for Iteration 1, will be part of your cuLearn submission. It will require you to supply the name of your test procedures for all functional items described above. Omitting to fill out this grid will result in a mark of 0: we will NOT study your code to try to find what you achieved…
You will also need to submit one or more snapshots demonstrating your tests successfully run.
Jars, source code and documentation will also be part of your cuLearn submission.