**Peer-Review #2:**

*Pietro Roggero, Lorenzo Tallarico, Matteo Trombetta, Tommaso Razza* |***Gruppo GC06****.*

Attualmente abbiamo scelto di implementare la comunicazione tramite **RMI**. I messaggi inviati, che abbiamo definito come “***Action***”, sono tutte sottoclassi della omonima classe e fanno tutti riferimento al GameController, il quale si occuperà di gestire una singola partita (dato che non implementeremo la funzionalità per partite multiple) che quindi gestirà la creazione e l’entrata nella partita.

Nel nostro progetto gestiamo la comunicazione tra client e server servendoci di due Blocking Queue: **serverActions** e **clientActions**. Queste code vengono gestite tramite l’utilizzo di due thread e gli elementi (Action) presenti in coda vengono smaltiti in maniera sequenziale modifiche inconsistenti al model da parte del controller.

In ***serverActions*** sarà il server ad aggiungere istanze delle sottoclassi di Action tramite un metodo che verrà invocato da parte del client dopo l’invio di un determinato comando. L’andamento sarà quindi questo:  
si avrà l’invio al server di un comando da parte del client, il comando verrà eseguito dal server il quale mediante un metodo aggiungerà alla coda serverActions una istanza di un oggetto sottoclasse di Action specifica per il comando che è stato inviato. Il server avrà un metodo che si occuperà continuamente di monitorare la coda, questo metodo non appena noterà l’aggiunta di un elemento sulla coda lo prenderà rimuovendolo da questa. L’oggetto ottenuto verrà processato in maniera differente a seconda del suo tipo e verrà invocato il controller che chiamerà un metodo (differente in base al tipo di Action) che a sua volta invocherà un metodo del model che attuerà la modifica.

Dall’altra parte alla fine di ogni metodo che attua una modifica nel model si avrà una chiamata ad un ***listener*** definito nella classe stessa del model. Il listener viene definito tramite una propria classe, avrà come attributo la lista dei client e implementerà tutti i metodi del tipo *notify...Action…* che vengono chiamati all’interno del model.   
Il listener mediante i metodi ***notify*** manda delle istanze di sottoclassi di Action ai client (ciascuna specifica per il tipo di modifica che è avvenuta sul model), il client a sua volta processerà le azioni ricevute in base al tipo e gli verrà mostrato l’update corrispondente. Tramite queste azioni verrà gestita anche lo stato in cui ogni player si trova che determinerà di conseguenza quali azioni potrà svolgere e quali no.  
La coda ***clientActions*** entrerà in gioco solamente per alcune operazioni che riguardano il comunicare determinati messaggi di errore, per la gestione della chat di gioco e peer la gestione delle connessioni al server. Il suo funzionamento è analogo a quello spiegato per serverActions.   
Concettualmente inviamo a tutti i client delle notify ogni volta che viene attuata una modifica al model, lato client avverrà poi un filtraggio solo delle informazioni che ciascun client deve ricevere a seguito di modifiche del model.

Riportiamo per completezza dei ***Sequence Diagram*** con degli esempi di interazione Client-Server:



