Peer Review

L'idea di base del nostro protocollo di comunicazione è quella di **mantenere sincronizzati**, in modo quasi speculare, il server e i client collegati alla partita. Il nostro client infatti possiederà un modello della partita leggermente modificato e con logica ridotta (*fat client*).

Dato che vorremmo gestire un server multipartita abbiamo deciso di creare un sistema che raccoglie i giocatori in **lobbies**, che consistono in sale d'attesa che si avviano quando tutti i giocatori sono collegati.

Essendo multipartita abbiamo predisposto il server in modo che sia possibile **risalire, dal nome del player,** al modello della partita a cui partecipa, alla lobby in cui è in attesa e ai nomi (e i socket) degli avversari della partita a cui il player sta partecipando.

Implementazione

Il protocollo sarebbe dunque a messaggi inviati attraverso una rete TCP. I messaggi sono stati modellati come sotto-classi che estendono la classe base ***Action***. Essi vengono serializzati e inviati via rete utilizzando la libreria **Gson** di Google, che serializza le classi atttraverso l'uso di file JSON.

Queste sono de-serializzate dal ricevitore all'interno della sottoclasse *Action*, corretta grazie a un attributo enum che ne indica il tipo.

Grazie alla divisione in sotto-classi i messaggi possono contenere informazioni di vario tipo, specifiche per l'azione stessa. Tutte le azioni contengono, oltre al tipo, anche il nome del player a cui l'azione è riferita.

Il controller del client o del server, ricevuta un'azione, potrà chiamare dei **servizi** specifici dell'azione passandola come argomento; essi faranno controlli sull'azione e chiameranno le funzioni di logica di gioco.

Messaggi

I messaggi si suddividono in due principali categorie:

- MenuAction
- PlayAction

Le action di menu sono messaggi relativi alle azioni svolte nel client al di fuori di una partita. Comprendono azioni di servizio come il Login, GetAllLobbies ecc... Vengono ignorate se mandate da un player che è già all'interno di una partita.

Le action di gioco sono azioni relative a mosse effettuate in partita. Per questo hanno senso solamente se un giocatore sta effettivamente partecipando in una partita. Le azioni di gioco vengono dunque ignorate se mandate da un player che non è in partita.

Protocollo

Abbiamo cercato di mantenere lo scambio di messaggi abbastanza simmetrico da entrambi i lati, utilizzando una logica a *servizi* sia nel server che nel client. In questo modo il client e il server utilizzano lo stesso pattern ed è possibile riutilizzare la maggior parte dei messaggi scambiati.

I messaggi saranno gestiti dal controller di entrambe le parti:

- Nel server la logica di gestione effettuerà molti controlli di validità dell'azione per poi effettuare le chiamate sul modello della partita. Fatte le opportune modifiche al modello i servizi del server recupereranno le informazioni necessarie alla risposta e le invieranno ai client interessati, che potrebbero essere solo il client che ha mandato il messaggio (come per esempio nel caso di login) oppure tutti i client connessi a una partita (come nelle azioni di gioco, il modello deve essere aggiornato su tutti i client connessi).
- Nel client invece il controller si occupa solo di deserializzare l'azione ed eseguire le chiamate necessarie ai metodi del modello, in modo da eseguire le stesse azioni ed essere sincronizzato con esso.

Alleghiamo qui un sequence diagram che raccoglie tutte le richieste che possono essere effettuate dal client e le risposte che invia il server.

Specifichiamo che le richieste non sono per forza in ordine e che le risposte saranno poi gestite in modo **asincrono** dal client, in quanto la ricezione dei messaggi dalla rete è gestita anche sul client in modo asincrono su un thread dedicato.