Peer-Review 2: Sequence Diagrams

D'Alessio Edoardo, De Ciechi Samuele, Deidier Simone, Ermacora Iacopo
Gruppo AM-41

Valutazione del diagramma UML delle classi del gruppo AM-04.

Lati positivi

- È molto interessante la vostra scelta di creare un unico messaggio da parte del server (*AckMessage(boolean)*) per notificare il client se l'azione richiesta è andata a buon fine o meno; questa scelta snellisce decisamente il numero di messaggi che il client deve poter gestire.
- Nella fase relativa alla prima connessione di un utente pensiamo abbiate gestito molto bene la differenza tra il giocatore che vuole creare la lobby piuttosto che colui che vuole unirsi ad una lobby già creata.

Lati negativi

- Nella fase relativa alla prima connessione di un utente il check sul nickname potrebbe essere fatto solamente per chi decide di *joinare* una lobby e "*farlo localmente a quella lobby*", alla fine avere due giocatori con lo stesso nickname in due lobby diverse potrebbe essere fattibile (*non implementando noi il multipartita non ci siamo mai posti questo dubbio*).
- Nella fase relativa alle dinamiche di gioco pensiamo abbiate inserito eccessivi controlli sugli input, i quali vanno un po' a saturare la connessione; si potrebbe pensare di accorpare i quattro messaggi in uno unico (*es.: Mossa*), il quale contiene tutte le informazioni (*tessere, ordine e colonna*). Per adattare la vostra idea di messaggio unico da parte del server, si potrebbe pensare di adattare *AckMessage(boolean)* come *AckMessage(boolean, int)*, dove in caso di *false*, *int* va ad indicare se il check è fallito per colpa di una colonna non utilizzabile (*non ci sono abbastanza slot*) o tessere non disponibili (*le tessere scelte non possono essere prese*). Inoltre, pensiamo che il check sull'ordine di inserimento dei *tiles* sia inutile, dato che non va a modificarne il numero (*se ha già passato i check precedenti, che l'ordine sia 1-2-3, 1-3-2 o altri non influisce*).

- Nella fase di gestione delle disconnessioni pensiamo che sia poco chiaro come gestiate il check sull'*ack*; ogni x secondi il server manda il messaggio di *ping*, al quale si aspetta un *ack*; il server assume il giocatore disconnesso dopo che y secondi dal messaggio di *ping* non è arrivata risposta, oppure quando invia un messaggio di *ping* dopo uno che non ha mai ricevuto risposta? Nel secondo caso, consigliamo di aumentare il numero di *ping* non risposti dopo il quale il server assume il player disconnesso (*noi abbiamo scelto 5*), perché magari per qualche ritardo di rete un giocatore correttamente connesso non riesce a rispondere ad un *ping*.

Confronto tra le architetture

- Avete deciso di non mandare un messaggio iniziale per fare il check sulla connessione dei giocatori, ma partire direttamente con un unico messaggio che esegue quello ed inoltre invia già il *nickname*; ci sembra un'ottima decisione che snellisce la parte di rete, la quale ci ha dato spunti interessanti per modificare la nostra implementazione.
- L'idea di un unico messaggio da parte del server è davvero interessante, siamo davvero colpiti. Ragioneremo sicuramente se prendere questa strada per snellire il nostro protocollo di rete che risulta molto corposo.
- L'idea che ad un player per riconnettersi alla lobby basti rimandare un messaggio di *ack* dopo aver ricevuto il *ping* è davvero intelligente: non sono necessari messaggi aggiuntivi (*es.: noi abbiamo creato il messaggio di Re-Join*) e questa meccanica non ha bisogno di una implementazione diversa dalla implementazione base di *ping-ack*; ci ha fatto riflettere molto sulla nostra decisone, in fase di scrittura del codice rivedremo come migliorare il nostro protocollo.