Peer review 2

Paolo Pertino, Leonardo Pesce, Alberto Paddeu

7 Maggio 2022

Valutazione design UML della parte di rete del gruppo G13

1 Lati positivi

Di seguito elenchiamo gli aspetti positivi:

- **Suddivisione dei messaggi**

  Notiamo dall'UML della parte di rete che i messaggi sono stati suddivisi in richieste e risposte: questo approccio lo troviamo funzionale perché in questo modo i messaggi sono ben suddivisi e facilmente gestibili.

- **Reduced Game**

  Nel metodo *VirtualView* abbiamo notato la presenza di una classe *ReducedGame*. Nonostante la classe non sia ancora specificata (nell'UML), riteniamo che l'utilizzo di una versione ridotta dell'oggetto *Game* sia una buona idea da continuare a implementare perché semplifica la gestione client-server del gioco, in quanto le azioni compiute in partita generalmente si basano sulla modifica di numerose parti del modello, e il pacchetto da inviare è di dimensioni abbastanza sostenute.

2 Lati negativi

Di seguito elenchiamo gli aspetti che secondo noi sono da migliorare:

- **Contenuto dei messaggi**

  Sarebbe stato interessante poter analizzare il contenuto dei messaggi e i parametri passati durante le comunicazioni, per comprendere le effettive dinamiche del protocollo e le interazioni con il controller(oggetti, interi, oggetti ridotti, stringhe...).

- **Gestione condizioni eccezionali**

  Riteniamo sia necessario inserire messaggi specifici che verifichino lo status della connessione tra client e server e che permettano da entrambi i lati di rilevare e gestire eventi eccezionali come disconnessioni impreviste dalla rete.

3 Confronto tra le architetture

Di seguito elenchiamo le differenze riscontrate tra la nostra architettura di rete e la loro:

- **Differenze nel MVC**

  Notiamo un differente approccio nell'utilizzo del pattern MVC: mentre nella nostra architettura le modifiche del modello di gioco sono notificate direttamente dall'oggetto *Game* alla *VirtualView*, nella loro il tutto viene effettuato dal controller che sembrerebbe creare lui stesso la versione ridotta del *Game* da notificare alla RemoteView e quindi da inviare al client.

- **Serializzazione** A differenza dei nostri colleghi, abbiamo preferito affidarci direttamente alla serializzazione di Java, vista la struttura e il contenuto dei nostri messaggi. I nostri colleghi a differenza nostra, come ipotizziamo guardando il loro UML(ad esempio l'oggetto *Parser* con i metodi *Encode* e *Decode*), sembra abbiano deciso di implementare la serializzazione utilizzando JSON.

- **Differente gestione del tipo dei messaggi client-server**

  Nella nostra architettura i differenti tipi di messaggi sono raggruppati in un'enumerazione; ogni messaggio è un'istanza di un oggetto *CommunicationMessage* distinguibile attraverso il tipo assegnatogli. I nostri colleghi invece hanno differenziato le varie tipologie di messaggi con oggetti distinti.

  Notiamo che questo loro approccio è interessante perché si possono avere messaggi più customizzabili e specifici per le varie fasi.