Peer-Review 1: UML

Poidomani Davide, Riboni Andrea, Volpentesta Edoardo
Gruppo GC51

8 maggio 2022

Valutazione del protocollo di rete del gruppo GC61.

1 Lati positivi

- Il client ha libertà di sbagliare: qualora inviasse una fase di gioco errata verrebbe notificato con un messaggio di errore (si suppone) adeguato
- Il ruolo del server come orchestrator e (quello che sembra essere) un thin client
- L'utilizzo dell'ereditarietà per la costruzione dei messaggi client-server
- La differenziazione a livello di classe di request e response

2 Lati negativi

- Come descritto nella relazione mandataci, concordiamo sul fatto che (per quanto semplifichi molto le cose) sia pesante inviare tutto il modello ad ogni aggiornamento
- C'è ridondanza nei messaggi
  - **Fase di setup**. il client si connette ed effettua la *SetupConnection*, dopodiché il server -qualora ad essersi connesso fosse stato il primo client- richiede di inserire il numero di giocatori: trovo che sia una richiesta superflua e sia integrabile direttamente nella fase

di setup e, al più, ignorata dal server (con comunque una relativa response ok/ko) qualora il client non fosse il primo

- **Ready**. Non è chiaro a cosa possa servire introdurre un'attesa della ricezione di un messaggio di *ready* da parte di ogni client: una volta inviata la *WizardChoice* sembra superfluo specificare anche *ReadyStatus*
- **Move (3) students**. Per questa fase di gioco vengono inviati 3 messaggi pressoché uguali consecutivamente. Sarebbe allora meglio
  * O aggregare queste tre richieste in una sola
  * O aggiungere, per ogni richiesta, una response di esito

3 Confronto tra le architetture

Il funzionamento del server della nostra architettura è paragonabile a quello preso in esame in questa review. La differenza sostanziale è data dalla gestione delle request/response: se GC61 ha preferito un approccio ad oggetti, differenziando ogni possibile messaggio; la nostra comunicazione si basa interamente sullo scambio di un oggetto *Action* serializzato, identificato da una *GamePhase* che ne identifica la fase e i conseguenti parametri necessari. Lo stesso oggetto viene utilizzato poi anche come response adottando due *GamePhase* "fasulle" (error, correct).

Per quanto troviamo più semplice lo scambio di un oggetto sempre *prefabbricato* nella stessa maniera, troviamo elegante la soluzione ad oggetti proposta da GC61.