Simone Rodari, Cristian Summa, Mathias Rodigari, Gabriele Pignataro
Gruppo GC42
Valutazione del diagramma UML del gruppo GC32

Peer Review

Lati Positivi:

- Buona l'idea di aver astratto l'implementazione di rete, in modo che sia Model che View possano operare senza alcuna modifica in entrambe le modalità.
- Idea interessante quella di implementare una chat privata tra due giocatori.
- Ottimo anche l'aver riunito tutto il codice dei "messaggi di aggiornamento" all'interno di *ListenersHandler*: in questo modo il *Model* presenta poche "interferenze" legate alla parte di Rete, e tutto il codice è invece facilmente localizzabile nelle due classi dedicate.

Lati Negativi:

- Suggeriamo l'utilizzo di una BlockingQueue invece di una semplice lista per salvare i comandi da eseguire. La BlockingQueue fornisce un meccanismo integrato per la gestione concorrente tra i thread, evitando problemi di sincronizzazione.
- Suggeriamo, per ragioni di manutenibilità del codice, di ridurre dove possibile il numero di metodi usati per la comunicazione nelle varie interfacce: ad esempio metodi come *sendResourceDrawn*() e *sendGoldDrawn()* potrebbero essere accorpati, e lo stesso potrebbe avvenire per tutti i metodi che trasportano "notifiche": accorpare la loro implementazione potrebbe semplificare non poco eventuali modifiche future, *bug fix* inclusi.

Confronto tra le architetture:

- Constatiamo che le due implementazioni della parte di Rete sono incredibilmente simili, almeno in termini di funzionamento: le interfacce *GameListener*, *NotifierInterface* e *ServerInterface* trovano tutte un'equivalente controparte nella nostra implementazione, con gli stessi scopi e quasi sempre le stesse funzionalità: interpretiamo la cosa come una conferma del fatto che entrambi siamo sulla giusta strada.
- Ci pare di aver capito che, nella vostra implementazione, il *Client* venga salvato nel campo *listener* di *RMINotifier*, e poi passato come parametro (quindi trasferito tramite la rete) sul *Server*, andando a finire all'interno della *listenersMap* contenuta nel *Game* (dentro *GameController*): se è così, allora anche qui è presente una somiglianza con la nostra implementazione, tuttavia ci sorge un dubbio: se il *Client* (o chi lo contiene) viene inviato come parametro, *RMI* lo invia *by value*, ovvero creandone una copia, e quindi chiamare poi i metodi su questa copia non produce effetti sul *Client* originale. Se, invece, siete riusciti a farlo funzionare senza dover ricorrere al *registry* anche lato *Client*, saltate questo punto.
- Anche lato *Socket* troviamo alcune somiglianze: il fatto di tenere il *Client* in *SocketClient* e di utilizzare invece la componente *Server* come sostituto di *RmiNotifier*, ad esempio, oppure la scelta di differenziare i messaggi usati per le due direzioni di comunicazione.
  Avremmo gradito vedere anche la struttura dei *Messages*, per capire come è organizzato lo scambio di informazioni, tuttavia dal numero elevato di metodi scritti presumiamo che anche voi, come anche noi del resto, abbiate preferito trasferire poche informazioni, il più semplici possibili, piuttosto che prendere la strada della *Serializzazione* degli oggetti.