**Peer-Review 1: UML**

Marco Andrea Brambillasca, Filippo Bertossi, Daniele Di Giorgio, Francesco Di Giore

Gruppo AM 40

Valutazione del diagramma UML delle classi del gruppo AM 03

**Lati positivi**

* Presenza di annotazioni utili alla comprensione e di uno schema del modello client-server (scelta di thick client).
* Utilizzo di una lista ordinata per il salvataggio del completamento dei Common Goal.
* Uso del Design Pattern Strategy, sia per i Common Goal che per la scelta dinamica dell’uso di RMI o TCP.
* La scelta di avere due sottoclassi LivingRoom e Shelf Controller permette di dividere la gestione di metodi che riguardano la board e le shelves.
* Utilizzo degli Observer per notificare la view delle modifiche sul model.
* Presenza nell’UML della specificazione della classe Messaggio nei vari tipi di messaggi (ben specificati)
* Utilizzo di molte classi astratte per una struttura molto flessibile

**Lati negativi**

Il messaggio di errata selezione viene inviato solo al termine della selezione stessa. Consigliamo di verificare subito se le coordinate passate (coppie di integer) corrispondono a una casella con una tile selezionabile.

Consigliamo il salvataggio delle Tiles selezionate dal player lato server per garantire la persistenza nel caso in cui il client si disconnetta dopo aver aggiornato lo stato del player a “CHOOSE\_COLUMN”. UpdateBoard è già stato chiamato e ha rimosso dalla LivingRoom le tiles selezionate in precedenza (che però non son state salvate).

La scelta della soluzione thick client non rende molto agevole l’aggiornamento della logica di gioco (i client dovrebbero fare aggiornamenti dedicati). Inoltre vi è un elevato rischio di stato inconsistente nel caso cada la rete durante il dialogo fra model e controller (esempio di sopra)

Non è chiaro come venga istanziato l’attributo personalGoal della classe PersonalGoal. Probabilmente le informazioni sui tiles da controllare per ogni personalGoal sono all’interno del codice, e personalGoal viene istanziata a inizio gioco. In tal caso, una modifica o un’aggiunta di nuovi PersonalGoal implicherebbe una modifica del codice, attività meno agevole di una variazione del contenuto di un file (es JSON).

Utilizzo di un gran numero di matrici. Forse si potrebbe optare per mappe, in quanto queste strutture dati espongono metodi di utility che rendono più leggibile il codice.

Consigliamo di creare una classe Bag dedicata, al posto di tenerla come all’interno della LivingRoom. In questo modo si potranno dividere i compiti di LivingRoom e Bag (nella quale si potranno raccogliere tutti i metodi da usare per il refill, il riempimento e lo shuffle delle tiles).

**Confronto tra le architetture**

I punti di forza della loro architettura sono la possibilità di un efficente dialogo fra Controller e View, entrambi sul client. Molto interessante è la scelta del pattern Strategy per la scelta dinamica del tipo di connessione (RMI o TCP).

Entrambi utilizziamo Observer nella view per notificare modifiche del Model.

Entrambi i team (ad oggi) usano 12 sottoclassi per la gestione dei CommonGoal. Si potrebbe considerare di trovare controlli comuni riutilizzabili in più CommonGoal, in modo da non avere 12 check specifici con grandi parti di codice ridondante.