Peer-Review 1: UML

Luca Bancale, Francesco Ambesi Impiombato, Riccardo Maria Caiani, Erika Airaghi

Gruppo AM24

Valutazione del diagramma UML delle classi del gruppo **AM33**.

# Lati positivi

L’aspetto che abbiamo apprezzato è stato il modo in cui viene gestita la lobby di gioco per l’opzione multi-partita.

# Lati negativi

Guardano il vostro UML sembrano mancare alcuni metodi e lo stato di varie classi, più nello specifico:

1. La classe *game handler* non può compiere alcuna azione su *game*.
2. Invocando i metodi della classe *Board* è possibile togliere i *Tile*, ma non si possono ottenere informazioni sullo stato del gioco*.*
3. *TilesBag* dovrebbe contenere una lista di *Tile*, per, tra le altre cose, facilitare il controllo che consente di verificare di non aver preso più di 22 tile per tipo; implementando la lista verrebbe sfruttato a pieno il metodo *RestoreTile(Tile).*
4. La classe *Shelf* andrebbe ampliata inserendo nuovi metodi, per esempio aggiungendo un metodo che controlli che la bookshelf sia piena, dato che non c’è la matrice di Tile (per esempio)*.*
5. Gli attributi booleani *inGame* e *inLobby* nella classe *User* potrebbero far parte di un’unica enum (così si evita di dover controllare che *inGame* e *inLobby* nonsiano entrambi settati per sbaglio come *true* contemporaneamente).
6. Inoltre avete de gli attributi *inGame* e *inLobby* sono duplicati in *User.*
7. Nella classe *Game* non si capisce bene come funziona *pickTiles(),* state comprimendo le coordinate in un unico int? Non è molto auto-esplicativa e semplice da gestire come modalità di procedere, e inoltre è molto error prone; sarebbe meglio creare una classe *Helper* per gestire le coordinate se le volete passare così. Stessa cosa in *Board* con *removeTiles()*.
8. *Player* non ha metodi per gestire lo stato, essendo tutto privato.
9. *GameHandlerInterface* dovrebbe avere tutti i metodi di *GamesHandler* (immagino l’abbiate creata per gestire il server RMI, se mancano i metodi nell’interfaccia poi non potete chiamarli).
10. Ritornare boolean dalle funzioni tipo *pickTiles(…)* non vi serve, in Java ci sono le eccezioni per gestire se una funzione è stata eseguita correttamente o meno.
11. Nella classe *Board* ci sono metodi per inizializzare la plancia per 2 o 3 giocatori, ma non 4.
12. I metodi per calcolare lo score andrebbero inseriti in *Player*.
13. In *Game* il metodo p*ickNextPlayer()* per ora ha poco senso perché la classe non ha uno stato e il metodo non ritorna nulla.
14. *PersonalGoalDrawer* come garantisce di non prendere duplicati? Anche questa classe non ha uno stato.
15. Nella classe *Lobby* ci sono 2 *ID.*

Consiglio: il metodo *values()* dichiarato nell’enum *Tile* è già fornito da Java (Tile.*values()*) e non serve re-implementarlo

Alcune domande

La classe *Controller* è necessaria nonostante vi sia già una classe *GamesHandler*?

# Confronto tra le architetture

L’architettura del model è simile alla nostra, avendo un Game che gestisce tutta la logica di gioco.

Una differenza, invece, riguarda la scelta implementativa nei Common Goal: nell’UML analizzato si è scelto di scrivere 12 classi per i Personal Goal.