Peer-Review 1: UML

<Matteo Panarotto>, <Maria Sole Rossi>, <Cristian Pasinetti>, <Daniel Straface>

Gruppo <GC-48>

Valutazione del diagramma UML delle classi del gruppo <GC-38>.

# Lati positivi

* In “Game”, interessante la presenza dell’attributo booleano “gameOn” per tenere traccia dello stato della partita qualora si volesse implementare una funzionalità avanzata come la resilienza alle disconnessioni o la persistenza della partita.
* L’utilizzo di un metodo ricorsivo per controllare l’adiacenza delle tessere oggetto nella libreria del giocatore è un’ottima idea, meccanica di gioco da noi dimenticata.
* L’analisi della saturazione della libreria mediante il metodo checkLastLine è ben congeniata in quanto sfrutta il controllo della singola riga utile anziché dell’intera libreria.
* Tutto ciò che non è indicato esplicitamente nei lati negativi o nel confronto ha una logica molto simile a quanto da noi utilizzato, efficiente e di facile comprensione.

# Lati negativi

* In generale, diversi metodi nel modello possono essere gestiti da entità presenti nella sezione Controller del pattern MVC, quali:
* In “Game”, la sequenza di mosse del giocatore durante il proprio turno, l’aggiornamento e il passaggio del turno fra i giocatori, dinamica della gestione del punteggio per ogni giocatore;
* In “Slot” la gestione di “Colore” e “Tipo” è ridondante e pleonastica: la definizione della colorazione della tessera oggetto ha solo un fine estetico e non funzionale circa il susseguirsi delle azioni di gioco. Sarebbe preferibile inserire questa particolarità solo quale forma di decoro nella sezione View del pattern MVC. Al contrario, la definizione del tipo concorre in maniera significativa circa i controlli e le scelte del giocatore. Indicare mediante nomi propri il tipo di Slot (quali gatto, libro, ecc…) consegna un’idea e una forma migliore sia allo sviluppatore del codice che all’utente.
* In “PersonalShelf” non troviamo una funzionalità utile al metodo “getSingleSlot” dal momento che ci sembra utile restituire l’intera libreria anziché la singola tessera oggetto.
* In “PersonalGoal” per la parte sul colore si rimanda a quanto detto prima.
* Per il cambiamento di dimensioni della plancia soggiorno in funzione del numero di giocatori della partita l’utilizzo di due colori fittizi (BLACK, GRAY) per identificare rispettivamente le posizioni sempre non giocabili e quelle vuote puo’ generare complessità e confusione.
* Nel diagramma UML:
* Vi è una mancanza d’indicazioni di cardinalità sul diagramma.
* Preferibile inserire commenti/spiegazioni delle entità utilizzando le apposite celle “note” anziché lasciarli liberi.
* Alcune relazioni fra classi sono poco chiare e intuibili (Esempi: composizioni non usate da PersonalShelf e PersonalGoal a Player, da Bag a Game).

# Confronto tra le architetture

* Riguardo l’implementazione del giocatore, i metodi preposti all’ordinamento ed inserimento delle tessere oggetto nella libreria personale del giocatore, il nostro gruppo ha deciso di formularlo lasciando nel model i metodi per l’estrazione e l’inserimento delle tessere mentre l’ordinamento è lasciato a carico della view, al contrario il gruppo revisionato ha voluto inserirli tutti nel modello. Inoltre è stato gestito mediante overloading il numero di tessere da inserire in contrapposizione alla nostra gestione mediante list di tessere oggetto siccome la sua dimensione è variabile.
* In “PersonalShelf” vi è dichiarato un booleano “itsFull”, un metodo “isItFull” e il metodo “checkLastLine” che di fatto eseguono la stessa funzionalità, ossia il controllo dell’ultimo turno di gioco. Noi abbiam accorpato il tutto in un unico metodo “isFull”, che ritorna vero se la libreria è piena, questo metodo viene poi chiamato dal controller nei controlli che si svolgono a fine turno.
* Per ottenere le coordinate relative alle tessere oggetto che costituiscono l’obiettivo della carta obiettivo personale è stato implementato da noi una lettura da file .json con relativa scrittura negli attributi della classe che rappresenta le carte stesse (il controller si occupa poi di distribuirli ai giocatori), mentre il gruppo revisionato ha scritto la funzionalità basandosi su metodi get/set e due variabili identificanti le coordinate.
* Come indicato nei lati negativi circa l’utilizzo di “BLACK” e “GRAY”, noi abbiamo utilizzato una matrice di validità di tipo intero che svolge in modo analogo la stessa funzione dei colori fittizi.
* Il controllo sul ripopolamento della plancia soggiorno e il metodo che effettivamente esegue il comando sono stati messi rispettivamente nel Controller e in Game, mentre l’altro gruppo li ha inseriti nella classe relativa alla plancia.
* Anziché implementare un metodo che modifica i punti da assegnare ad un giocatore, abbiamo utilizzato un array dichiarato nella classe della carta obiettivo comune che contiene i punti da assegnare in base al numero di giocatori presenti (la sua inizializzazione cambia di partita in partita) e verrà decrementato qualora un giocatore ottenga un obiettivo.
* Siamo riusciti ad abbassare il numero di sottoclassi utilizzati per i controlli delle carte obiettivo comune arrivando ad un minimo di 5: abbiamo accorpato controlli “simili” fra loro (quali le tre colonne e quattro righe formate da tessere con al massimo tre tipi diversi di oggetti) oltre a quelle individuate dal gruppo revisionato.