Peer-Review 1: UML

Laura Maria Bosco, Viola Caliceti, Tommaso Andaloro, Andrea Ardu

Gruppo 43

Valutazione del diagramma UML delle classi del gruppo 06.

Lati positivi

- Utilizzo della lobby per la gestione della vita del player e creazione partita
- Utilizzo di una classe CommonObjective/PrivateObject che va a chiamare il metodo checkCondition() nelle sue sottoclassi
- Utilizzo della classe Matrix sia nella classe BookShelf, sia in GameBoard, con i relativi metodi di inserimento delle Tiles separati
- Utilizzo della classe predefinita di Java Point, piuttosto di crearne una nuova appositamente per il gioco
- utilizzo di molti metodi essenziali, necessari per evitare l'uso di metodi troppo complessi
- implementazione di un uml già rivolto ad alcune specifiche bonus, come chat e multipartita
- implementazione del server e controller schematici e intuitivi

Lati negativi

- nella classe Board alcuni metodi risultano poco chiari, per esempio boardResetITA(), checkFreeMOSTCOMPLICATED(), freeTilesMOSTCOMPLICATED(), boardResetENG()
- Le carte CommonObjective possono essere raggruppate in delle classi in base alla loro somiglianza: così si evita di avere 12 sottoclassi diverse e si diminuiscono le righe di codice da scrivere
- Le carte PrivateObjective funzionano tutte con la stessa logica, si potrebbe quindi fare un'unica classe a cui viene passato un file json con tutte le configurazioni possibili
- Nella classe Sachet si potrebbe utilizzare un unico metodo removeTiles con parametro una lista di Tiles, da cui si può ricavarne il numero col metodo .size()
- Mancano i metodi getter dei parametri numRows e numCols della classe Matrix
- La classe Player ha come attributi vari tipi di punteggi che derivano da vari obiettivi, ma si potrebbe tener conto del punto del giocatore in un'unica variabile
- Non è chiaro come venga utilizzata la classe MoveNotPossible; in che modo il controller andrà a chiamare i metodi (assenti) di quest'ultima nel momento in cui dovrà controllare la validità della mossa?

- nel caso di personalObjective si potrebbe lavorare tramite un unico metodo checkCondition sulla bookshelf, per poi nel caso assegnare al player i personalObjectivePoint, senza dover fare il doppio check sul player tramite checkCondition e senza ripetere il controllo di personalObjectivePoint sempre su bookshelf. (ovviamente nel caso in cui checkCondition e personaleObjectivePoint svolgano istruzioni molto differenti in base al parametro passato).
- IntelliJ, creando l'uml non ha collegato tutti gli elementi (come clientBase, gameState) e la grafica è un pò confusa (comunque comprensibile)
- poco chiaro Tiles e i suoi metodi riguardanti il valore (immaginiamo tiles sia la pedina?)

Confronto tra le architetture

- Il nostro Model, in modo simile al vostro, si appoggia su una classe principale GameModel, che si occupa della gestione della partita, utenti e contiene attributi e le componenti del gioco.
- é stata implementata in modo diverso la Board: avevamo considerato in precedenza come voi l'utilizzo di una matrice, ma poi abbiamo optato per mappare tramite un numero ciascuna casella e il suo contenuto assieme agli indici dei vicini di questa.
- Come detto in precedenza, abbiamo implementato in modo diverso le classi riguardanti le tipologie di common objective Card e private Objective Card, cercando di riutilizzare più codice possibile. Dal documento da voi inoltrato, capiamo la vostra scelta, anche se potreste avere maggiori casi di test da implementare.
- per controllare le mosse, abbiamo pensato a 2 classi Move e MoveController, situate nel controller, mentre dal vostro Uml sembrerebbe che il controllo avvenga sul Model.