Peer-Review 1: UML

Saccani Federico, Spangaro Francesco, Sanvito Luca, Pedersoli Luca

Gruppo https://github.com/TheFedelino01/ing-sw-2023-saccani-spangaro-sanvito-pedersoli

Valutazione del diagramma UML delle classi del gruppo
https://github.com/LucaParsani/ing-sw-2023-luisi-parsani-simoncini-zanardi.

Lati positivi

L'UML si presenta in una forma chiara di facile comprensione.
Il livello di dettaglio è sufficiente a capire le strada implementativa che si penserà di realizzare in futuro; i metodi di maggior interesse per le dinamiche di gioco sono presenti nel Unified Modeling Language.
L'introduzione del package "CommonGoals" rende immediata la suddivisione delle carte obiettivo comuni dal resto dell'implementazione: l'insieme è così già da subito diviso.
Molto apprezzata la scelta di introdurre il pattern Factory per il checking dei CommonGoal (anche se nel diagramma UML viene chiamato con il nome di Strategy).
Si nota la giusta cura al dettaglio richiesta per un primo approccio alla progettazione del Model.

Lati negativi

Di seguito vengono riportati alcuni suggerimenti in merito ad aspetti di possibile modifica o miglioramento:

- **"PersonalGoal"**
  - **Utile inserire una Enumeration "PersonalGoalType"per distinguere le differenti PersonalGoal**
    Così facendo la lista di UsedCodes non è più necessaria in quanto il controller, durante l'assegnamento delle carte obiettivo personali, dovrà solo guardare il tipo di PersonalGoalType capendo quali sole le carte già assegnate)

  - **Inserire un attributo per implementare la mappatura dell'assegnazione del punteggio**
    Le carte obiettivo personali restituiscono punti a seconda di una legenda in base a quanti match positivi si trovano in personalGoal[6][5]

- **"CommonGoal"**
  - **Utilizzare come per le PersonalGoal un Enumeration "CommonGoalType"**
    (come proposto per le PersonalGoal)

  - **Fusione tra le sottoclassi: Utilizzare meno sottoclassi per implementare il Pattern Strategy di CommonCard**
    Alcuni pattern da verificare con "check(Shelf shelf)" hanno caratteristiche in comune, quindi, è possibile unirle tra di loro ottenendo così non 11 sottoclassi ma un sottoinsieme più ristretto (circa 6).

  - **Cambiare il nome** all'interfaccia "CommonGoalStrategy" in "CommonGoalFactory"

- **“Game”**
  - **Inserire Eccezioni**
    È utile inserire delle eccezioni per far capire al controller eventuali errori che si verificano durante le operazioni (es. Player già esistente, Mossa non valida, etc.)

  - **Introdurre Stato del “Game”**
    Aggiungere un Enumerazione che permetta di capire in quale stato si trovi il Game. Cosí facendo, si può distinguere:
    - **Waiting:** un Game che deve aspettare l’ingresso di altri giocatori per iniziare (es. i Player sono ancora in Lobby);
    - **Running:** lo stato che permette di capire che il gioco è in corso e quindi non ammette nessun altro giocatore;
    - **Ended:** Game terminato.

- **“Player”**
  - “commonGoalRedeemed” ha un forte legame con la posizione degli elementi presenti nella lista dei CommonGoal nella classe Game: si potrebbe sostituire con una lista di “CommonGoalStrategy” (lista di interfacce)
    Così anche l’attributo passato nel metodo “isCommonGoalRedeemed(int indice)” sarà di tipo CommonGoalStrategy.

  - “tryPersonalGoal(int goalCode)”
    Potrebbe essere implementato dal Game in quanto è il game che possiede tutta la lista dei players e di conseguenza il controllo per capire se una PersonalGoal è stata già assegnata viene svolto semplicemente scorrendo una volta la lista dei players.
    (Cosí come è implementato, come fa il singolo Player ha conoscere le PersonalGoals degli altri players presenti?)

- **“Chat”**
  - **Riferimento al Player che invia il messaggio**
    Aggiungere una classe Messaggio che contiene attributo String msg e attributo Player sender cosí è possibile capire chi ha inviato un determinato messaggio

  - Si potrebbe aggiungere un attributo che rappresenta la data e ora di invio del messaggio

Confronto tra le architetture

La 2 architetture, in linea generale, hanno più o meno lo stesso concept di base.
Riportiamo di seguito alcuni delle differenze progettuali più significative:

Per alcuni metodi come: l’estrazione delle carte personali, l’estrazione delle carte obiettivo comuni, etc. il gruppo revisionato ha preferito implementarle direttamente a lato Model; il nostro gruppo, invece, le ha implementate a lato Controller in quanto ritenute non strettamente collegate al Game ma bensì ad una sua particolare implementazione.

Un altro aspetto differente risiede nella suddivisione delle sottoclassi per l’implementazione del controllo del checking degli obiettivi comuni: l’idea di base di dividere il controllo in sottoclassi è stata la stessa, la differenza è data dal numero di sottoclassi utilizzate (gruppo revisionato: 11, nostro gruppo: 6).

Un altro aspetto differente è il modo di gestire il controllo dei punti assegnati relativi alle carte comuni: si può ottenere il punto da una carta obiettivo solo una volta.

Il nostro gruppo utilizza la classe Point che memorizza anche l'informazione da quale carta Comune o Personale è stato generato (permettendo così il controllo), il gruppo revisionato basa il controllo su una Lista di boolean.
Probabilmente il checking utilizzando la lista di boolean avviene in modo più immediato in quanto basta accedere alla posizione della carta comune che si vuole verificare.
Con l'informazione memorizzata dentro al Point bisogna scorrere tutti i Point (al più 3) e verificare se un determinato obiettivo è stato soddisfatto oppure no.
La nostra scelta è stata dettata più per un livello stilistico in quanto pensiamo sia il Point a doversi ricordare da quale carta è stato generato e non il player. Inoltre, distinguendo il vettore di points di tipo int e la lista di booleani si perde il mapping di quale carta ha generato quale punto.
Nonostante questo, le 2 implementazioni sono essenzialmente identiche a patto di complessità lievemente differenti anche se costanti (theta(1) e O(3)).

Tutto sommato, l'UML del model non ha grosse differenze e entrambe le scelte implementative permettono di svolgere le operazioni fondamentali richieste nella specifica del gioco MyShelfie.