PeerReview IS23-AM45

Reviewer: AM36

1 April 2023

Indice

1 Analisi UML **3**
1.1 Comprensione UML 3
1.2 Mancata Comprensione UML 3

2 Analisi Tecnica **4**
2.1 Punti Positivi 4
2.2 Punti Negativi 4
2.2.1 Estendibilità 4
2.2.2 Riusabilità 4
2.2.3 Criticità Essenziali 5

3 Consigli **6**

4 Resoconto Finale **7**

1 Analisi UML

Di seguito viene riportata la nostra comprensione dell'UML fornitoci, quindi ciò che non è stato compreso affondo.

1.1 Comprensione UML

La struttura del model si basa praticamente sulla classe Lobby, la quale contiene un'istanza di Game con i relativi player e con la sua LivingRoom. Ogni mossa del giocatore passa attraverso il metodo move all'interno del game, che fa i controlli necessari per la correttezza della mossa. La LivingRoom è formata a sua volta da una matrice di Tile, il Player invece ha UN, points ecc... . Ogni Player quindi avendo la sua BookShelf e il suo personalGoal procede nel costruttore del Player stesso a istanziare le relative Classi al suo interno. I commonGoal sono modellati utilizzando un pattern di tipo Strategy, quindi istanziando 12 CommonGoal e poi chiamando il metodo all'interno del commonGoal all'interno del Game.

1.2 Mancata Comprensione UML

Non abbiamo trovato la classe ScoreToken all'interno dell'UML. Non abbiamo compreso affondo la scelta dell'utilizzo di una List di List di Optional di Tile come shelf in confronto con la scelta di utilizzo di una matrice di Tile per rappresentare quella che è la Board nella livingRoom, in quanto i due utilizzi sono molto simili e quindi potrebbero essere modellati nello stesso modo.

2 Analisi Tecnica

Di seguito sono riportati i punti di forza e di debolezza evidenziati svolgendo un'analisi funzionale sull'esendibilità, riusabilità e modificabilità della struttura compresa nell'UML.

2.1 Punti Positivi

- **Ottimo utilizzo della classe TileBag**: in quanto nel momento in cui si volesse modificare la logica dietro la distribuzione/estrazione delle carte si potrebbe semplicemente modificare la logica della classe interna.
- **Ottima strutturazione della classe PersonalGoal**.
- **Ottimo incapsulamento delle classi**: Un Game ha una livingRoom e dei player, i player hanno le Bookshelf e un personalGoal ecc...
- **Utilizzo intelligente di scoreTokens**: che presupponiamo serva per ricalcolare il punteggio del giocatore ogni volta che termina il turno.
- **Utilizzo Optional**: Ottima scelta implementativa in quanto l'utilizzo degli Optional consente maggiore robustezza contro le eccezzioni dovute all'accesso ad un campo null.

2.2 Punti Negativi

Non sappiamo se la versione dell'UML fornitoci sia l'ultima oppure una versione iniziale oramai obsoleta. Quindi ci limitiamo a riportare cosa, previa comprensione dell'UML, abbiamo evidenziato come criticità nella struttura del model.

2.2.1 Estendibilità

- **Argomenti di chiamata in Move e selectTiles**: modellare inserendo una serie di argomenti lega molto fortemente l'estendibilità del modello in quanto se dovessi poi scegliere che ogni Presa del giocatore può avere più di 3 Tiles dovrei andare a modificare la signature del metodo e quindi anche a modificare tutte le chiamate della funzione all'interno del codice già scritto.
- **Classe PersonalGoalBag poco estendibile**: il non avere una lista di PersonalGoal all'interno della PersonalGoalBag implica che se dovessi aggiungere un personalGoal in futuro dovrei aggiungere un altro campo pgN.
- **Modellazione CommonGoals**: la creazione delle 12 classi ovviamente non è una soluzione sbagliata ma come già detto dai prof, limita l'estendibilità, in quanto se volessi aggiungere altri commonGoal dovrei creare un'altra classe e poi istanziarla nel momento in cui devo scegliere quale commonGoal inserire nel Game, il che porterebbe ad uno spreco di spazio.
- **Modellazione Interna del Game**: Utilizzo di due attributi distinti per fare riferimento ai commonGoal, il che porta a difficoltà nel caso in cui in futuro decidessi di aggiungere un commonGoal in più per ogni partita (Argomento ripreso nei consigli).

2.2.2 Riusabilità

- **Granularità dei metodi**: Da cio che si evince anche dalla descrizione fornita, tutto il gioco si centra sul metodo move, il che porta a non poter poi, se volessi aggiungere un metodo nuovo in altre classi, riutilizzare altri metodi già implementati e testati in modo da ridurre il tempo di sviluppo e la probabilità di presenza di bug.
- **Errato posizionamento dei metodi**: il posizionamento di getWinner() all'interno del player non ha nessun'attributo del tipo winner da ritornare nel caso in cui un giocatore dovesse risultare vincitore, se invece il funzionamento di getWinner() prevede prendere il giocatore con il più alto punteggio e ritornarlo, allora getWinner si trova nella posizione sbagliata. in quanto dovrebbe trovarsi dove l'attributo di riferimento è istanziato cioè nel Game.

2.2.3 Criticità Essenziali

- **Attesa Risposta Client**: Da come si evince nella descrizione fornita accanto all'UML, nel metodo move si aspetta una risposta del client, sospendendo in questo modo l'intero server o almeno l'intero thread che si occupa del player, quindi rendendo impossibili altre azioni al giocatore oltre allo specificare la colonna. Ciò è una scelta implementativa poco corretta perchè impedisce ad esempio al player di uscire dalla partita fino a che non ha scelto una colonna. E come hanno anticipato i prof a laboratorio il server non dovrebbe chiedere al client qualcosa che vuola ma dovrebbe essere il client a fornirgliela autonomamente(Argomento ripreso nei consigli).

3 Consigli

Di seguito si trovano dei consigli mirati all'aumentare l'estendibilità e la granularità del codice, che avremmo fatto noi, per da garantire un' elevata modificabilità e partizionamento del modello.

- **Utilizzo di Mappe come Board o Shelf**: Soprattutto per la board che è una matrice sparsa converrebbe utilizzare una mappa in quanto tale struttura possiede metodi molto comodi e utili per la sua stessa gestione, per quanto riguarda la shelf invece anche una semplice matrice andrebbe bene, perchè potendo essere riempita completamente per vedere se la shelf è piena, basterebbe controllare tutti gli elementi in riga 0.
- **Utilizzo di Liste per risolvere Estendibilità**: esempio ne è la classe PersonalGoalBag in quanto se tutti i personalGoal venissero inseriti in una lista, sarebbe anche più facile estrarne uno a caso semplicemente randomizzando un indice, e nel momento in cui si dovesse decidere di estendere i personalGoal con l'aggiunta di altri N, basterebbe aggiungerli alla lista e sorteggiare sulla lunghezza della stessa. Stesso vale per i commonGoal nel Game.
- **Utilizzo JSON**: come anticipato a Laboratorio dal ingConti sarebbe utile utilizzare i json in quanto verrebbe molto semplice la configurazione delle partite oppure della costruzione delle classi CommonGoal o PersonalGoal.
- **Rimozione di attributi ridondanti**: Noi avremmo rimosso **nPlayers** presente come lunghezza dell'array di Player, **lastRound** perchè è possibile controllare se la partita sia finita anche controllando che esiste almeno una libreria completa e che il giocatore non sia quello di turno = 0 in modulo players.lenght, **chair** all'interno di Player in quanto si può ricavare chi è il primo giocatore dal Game, e soprattutto **currentPlayer** in quanto è identificabile dal Player presente in posizione **round** dentro Players.
- **Aggiunta metodi utili**: metodi utili come metodi di controllo correttezza di gioco, separati dai metodi di gioco, ad esempio un metodo che ritorna quante posizioni libere ci sono in una colonna, che può essere quindi usato per controllare se la matrice è piena e controllare se il Player può inserire la sua presa in quella colonna, il che sarebbe una generalizzazione dell'attributo maxTilesPerMove.
- **Refactoring dei nomi di metodi e attributi**: rinominare gli attributi per una più chiara comprensione del loro significato e anche del loro utilizzo, quindi rinominare i metodi per evidenziare in maniera più efficiente la loro funzionalità.

4 Resoconto Finale

Il progetto in sè funziona a patto di aggiustare la criticità evidenziata precedentemente, manca laggermente di scalabilità ed estendibilità, ma dalla nostra analisi viene che l'idea dietro c'è ed è chiara, ma la sua modellazione pecca di ridondanza e troppa centralità del codice.
Speriamo di essere stati utili nel recensire il vostro lavoro, e che per qualsiasi eventuale perplessità o dubbio possiate voi rivolgervi a noi senza remora alcuna.
Grazie per la puntualità nel rispetto delle consegne.

Buon lavoro.

AM-36