Valutazione del diagramma UML delle classi del gruppo <09>.

1 Lati positivi

Uno dei punti positivi è la semplicità del diagramma. Per fare una veloce simulazione del gioco basta guardare i metodi di tre classi quindi è relativamente semplice da fare. Questa semplicità ha però parecchi lati negativi che, secondo noi, surclassano quelli positivi.

2 Lati negativi

Il diagramma ha due classi ("Controller" e sopratutto "Player") troppo dense di metodi ma al contempo mancano attributi importanti che permettano di incapsulare lo stato del gioco in run time.

Simulando il gioco immaginiamo che il client mandi delle richieste al server per modificare lo stato del gioco a partire dalla classe CONTROLLER.

1) Il client manda una richiesta "aggiungere i player" al server, viene invocato getPlayers(), da CONTROLLER, che dovrebbe ricevere in parametro almeno gli username, che istanzia al più quattro oggetti PLAYER. Perfetto.

2) Il client manda una richiesta "pescare e posizionare per la prima volta gli item tiles sulla SERVERBOARD" al server, viene invocato getBoard(), da CONTROLLER, che crea l'istanza dell'oggetto SERVERBOARD e invoca fill() da SERVERBOARD. Perfetto.

3) Il client manda una richiesta "pescare e piazzare sulla board le due common goal card" al server, viene invocato getGoals(), da CONTROLLER, e getCommonGoals() da SERVERBOARD. Immaginiamo che ci sia un generatore randomico di CommonGoal che istanzia una specifica CommonGoal partendo dall'oggetto astratto CommonGoal. Il codice potrebbe appesantire parecchio SERVERBOARD ma non è un punto critico.

4) Il client manda una richiesta "dare una bookshelf a tutti i player" al server, viene invocato getShelves(), da CONTROLLER, che a sua volta invoca getShelf() da PLAYER. Perfetto.

5) Il client manda una richiesta "pescare la personal goal card per ogni player" al server, non essendoci metodi ad hoc supponiamo che venga eseguita la pescata della personal goal card insieme alle common goal card, viene quindi invocato getGoal(), da CONTROLLER, si esegue quanto scritto sopra

e poi non è chiaro come si prosegua. Non ci sono metodi per istanziare la classe PLAYERGOAL direttamente da PLAYER quindi deve essere fatto tutto da CONTROLLER che diventa quindi ancora più pensate seguendo il ragionamento di prima.

6) Il client manda una richiesta "scegliere e memorizzare il first player seat" al server, viene invocato il metodo getCurrentPlayers() da CONTROLLER. Anche così non esiste da nessuna parte un attributo che mantenga in memoria il first seat player.

7) Il client manda una richiesta "iniziare il primo turno del gioco" al server, dal modello non è chiaro come venga gestita e controllata la turnistica del gioco, tutto questo lavoro sembra essere derogato al metodo moveTiles(Player player, Move move), da CONTROLLER, il che appesantirebbe ulteriormente la classe CONTROLLER. E' difficile commentare il resto in dettaglio perchè manca sullo stesso diagramma. Potrebbe essere invocato getCurrent()Player al posto di moveTiles() ma le conclusioni sono le stesse.

8) Il client manda una notifica "il Player ha pescato x item tiles dalla ServerShelf" al server, viene invocato il metodo moveTiles(Player player, Move move), da CONTROLLER. Non è chiaro cosa sia l'oggetto Move in parametro perchè non presente nel diagramma, immaginiamo che siano le posizioni delle tiles da pescare sulla SERVERBOARD e le posizioni e ordine di inserimento nella SERVERSHELF. Questo implica che in un solo "passo" è stato mandato un messaggio molto pesante con vari stati del gioco (pescata dei tiles dalla board e controllo di eventuali pescate non consentite, posizionamento dei tiles e controllo di eventuali inserimenti errati), sarebbe meglio "atomizzare" questo metodo in più metodi.

9) Il client manda una richiesta "controlla se il player ha raggiunto il common goal" al server, non ci sono metodi da invocare da controller quindi dovrà essere derogato di nuovo a moveTiles(), vale il ragionamento fatto sopra sulla eccessiva pesantezza e l'atomizzazione del metodo.

10) Non è chiaro il funzionamento del gioco una volta che un player ha riempito la bookshelf. Non ci sono metodi invocabili da CONTROLLER né per il controllo dell'ultimo turno né per il controllo dello score, immaginiamo che venga derogato anch'esso a moveTiles() (si ripete il ragionamento fatto sopra). Viene invocato hasFinished() da PLAYER alla fine del turno del player, e poi? Se si è all'ultimo turno e si deve controllare lo score chi lo fa? come? Dal diagramma sembra che sia player a controllare i common e personal Goal mentre il conto degli item adiacenti è fatto da SERVERSHELF. Sarebbe meglio avere una classe con visibilità sia sulle common goal cards che

sul player così che sia sufficiente invocare un metodo delle common goal card che controlla la logica della carta sulla SERVERSHELF passata in parametro. Non è chiaro chi faccia la comparazione degli score e indica il vincitore.

CONCLUSIONE LATI NEGATIVI: in sostanza manca un po' di ordine e ci sono troppi metodi che fanno troppe cose, sarebbe meglio migliorare il diagramma UML tenendo conto della leggibilità e sopratutto della visibilità delle varie classi l'una con l'altra per evitare codice ridondante e/o troppo pesante.

3 Confronto tra le architetture

Onestamente il diagramma che abbiamo ricevuto non può essere ritenuto completo e quindi confrontarlo con il nostro è difficile. E' stato difficile fare una valutazione date le poche informazioni e dettagli presenti nel diagramma, ma un primo confronto si può fare ad occhio notando che il nostro diagramma è strutturato per non avere troppi metodi pesanti in una sola classe mentre il diagramma del gruppo AM09 ha specialmente questo problema.