Peer-Review 1: UML

Leonardo Ratti, Francesco Rivitti, Denis Sanduleanu, Alessandro Salvatore

Gruppo 30.

Valutazione del diagramma UML delle classi del gruppo 20.

Lati positivi

Indicare in questa sezione quali sono secondo voi i lati positivi dell'UML dell'altro gruppo. Se avete qualche difficoltà, provate a simulare il gioco a mano, immaginandovi quali sono le invocazioni di metodo che avvengono in certe situazioni che vi sembrano importanti (ad esempio, la fusione delle isole oppure il calcolo dell'influenza).

- Giusta l'idea di inserire un'enumerazione per i tipi delle tiles.
- Buona l'idea di utilizzare un metodo per la Shelf che ritorna il massimo numero di tiles selezionabili in base alla disponibilità delle colonne (*) .
- Buona intuizione inizializzare la Board leggendo un file json. Questo fa in modo che la board possa essere modificata più facilmente in una eventuale nuova versione del gioco.

Lati negativi

Come nella sezione precedente, indicare quali sono secondo voi i lati negativi.

- Consiglio di rinominare la classe "Deck" con "Bag" (sacchetto), che rende più chiara l'idea dell'oggetto concreto che si vuole rappresentare. Potrebbe confondersi con un eventuale classe DeckPersonal e DeckCommon per contenere le carte personali e comuni.
- Consiglio di accorpare la classe Tile e la classe Couple in un'unica classe Tile con anche l'attributo "state". Nel caso in cui le tiles siano ancora nel Deck, impostare lo stato a "invalid".
- (*) Manca l'intera segnatura del metodo nella classe Shelf dell'UML.
- Non è presente nella classe Game di un attributo LivingRoom da passare per esempio al metodo refillLivingRoom(). Inoltre sarebbe più corretto spostare questo metodo all'interno della classe LivingRoom, così che sia più semplice accedere e modificare i suoi attributi per il refill.
- Non è ben chiaro come vengano gestiti i turni della partita. Ogni player corrente esegue una serie di operazioni che si ripetono in tutti i turni; i metodi in questione sono chooseTiles, chooseOrder, chooseColumn: una volta che il primo player esegue questi 3 metodi, come si passa al prossimo turno? (o meglio, come fa il Game a sapere che deve eseguire nextTurn?).

- Non comprendo il funzionamento della classe CommonGoalCard: ci sono 2 metodi che possono essere utili per le “parti algoritmiche” delle carte, ma non riesco ad evincere dall’UML la realizzazione delle carte vere e proprie.

- Le CommonGoalCard scelte per la partita giocata non sono salvate da nessuna parte: ogni partita ha la sua coppia di carte obiettivo comune che fa in tutto e per tutto parte del Model, quindi mi aspetto che vengano scelte e rappresentate nel gioco in qualche modo, per esempio salvarle in un array di CommonGoalCard in Game.

- Manca uno stato in cui le tiles sono pickable, ma allo stesso tempo non possono essere prese perché non hanno lati liberi (es. le tiles al centro della board). Ovviamente servirebbe anche un metodo che controlla quali tiles sono effettivamente accessibili e quali invece sono pickable ma non accessibili. Anziché usare State.Empty potreste usare null.

- L’oggetto Game ha solo un attributo List<Player> e un currentPlayer, penso che dovrebbe contenere tutte le classi che servono per svolgere una partita, anche in previsione di una futura implementazione per giocare più partite contemporaneamente.

Confronto tra le architetture

Individuate i punti di forza dell’architettura dell’altro gruppo rispetto alla vostra, e quali sono le modifiche che potete fare alla vostra architettura per migliorarla.

- Buona l’idea di utilizzare un metodo per la Shelf che ritorna il massimo numero di tiles selezionabili in base alla disponibilità delle colonne (*).

- Buona la creazione separata di un metodo che permetta al giocatore di scegliere l’ordine in cui il player vuole inserire le tessere nella shelf (chooseOrder()). Questa è un’aggiunta che dovremo fare anche al nostro progetto.