02 – Elaborazione - Iterazione 6

1 Introduzione

Durante questa sesta iterazione ci si concentrerà su:

- Implementazione dello scenario principale di successo del caso d'uso *UC6: Visualizza le regole del gioco*, in modo da permettere al giocatore di poter visualizzare le regole del gioco, selezionando la relativa voce dal menu delle opzioni presenti nel gioco.
- Analisi e progettazione dello scenario principale di successo del caso d'uso *UC7: Chiedere un suggerimento*, in modo da permettere al giocatore di visualizzare l'eventuale mossa consentita in quel momento, sempre secondo le regole del gioco e lo stato attuale dello stesso. Per questo caso d'uso in particolare non è stata fatta un'implementazione nel codice.
- Implementazione dei casi di test, utilizzando il framework jUnit, che permettono di rilevare eventuali malfunzionamenti del codice.

2 Presentazione del caso d'uso UC6

Andiamo nuovamente a presentare lo schema del caso d'uso *UC6: Visualizza le regole del gioco*

UC6: Visualizza le regole del gioco

1. Il giocatore desidera visualizzare informazioni aggiuntive sul gioco e clicca sul bottone "*Regole del gioco*".

2. Il programma visualizza una finestra contenente le regole base del solitario considerato.

2.1 Presentazione del caso d'uso UC7

Andiamo nuovamente a presentare lo schema del caso d'uso *UC7: Chiedere un suggerimento*

UC7: Chiedere un suggerimento

1. Il giocatore clicca sul tasto "Aiuto" o sul simbolo "?" presente nella barra del menù.

2. Il programma, sulla base delle mosse consentite in quel momento, visualizza sul tavolo da gioco due carte che possono essere selezionate e posizionate l'una sull'altra per continuare il gioco.

3 Analisi Orientata agli Oggetti

Al fine di descrivere il dominio da un punto di vista ad oggetti e gestire ulteriori requisiti, saranno utilizzati nuovamente gli stessi strumenti dell'iterazione precedente (Modello di Dominio, SSD Sequence System Diagram e Contratti delle operazioni). In particolare, i paragrafi seguenti permettono di evidenziare i cambiamenti che tali elaborati hanno subito rispetto alla fase precedente.

3.1 Modello di dominio

Dall'analisi dei casi d'uso UC6 e UC7, emerge la nuova classe concettuale **SolitaireRules**. Di conseguenza, viene qui sotto riportato il modello di dominio risultante.

Player: Rappresenta l'attore primario, che interagisce col sistema per eseguire le operazioni

SolitaireGame: Rappresenta l'applicazione del gioco Solitario

Deck: Rappresenta un mazzo di carte francesi costituito da 52 carte

Card: Rappresenta il concetto di carta da gioco, caratterizzata da un valore e da un seme

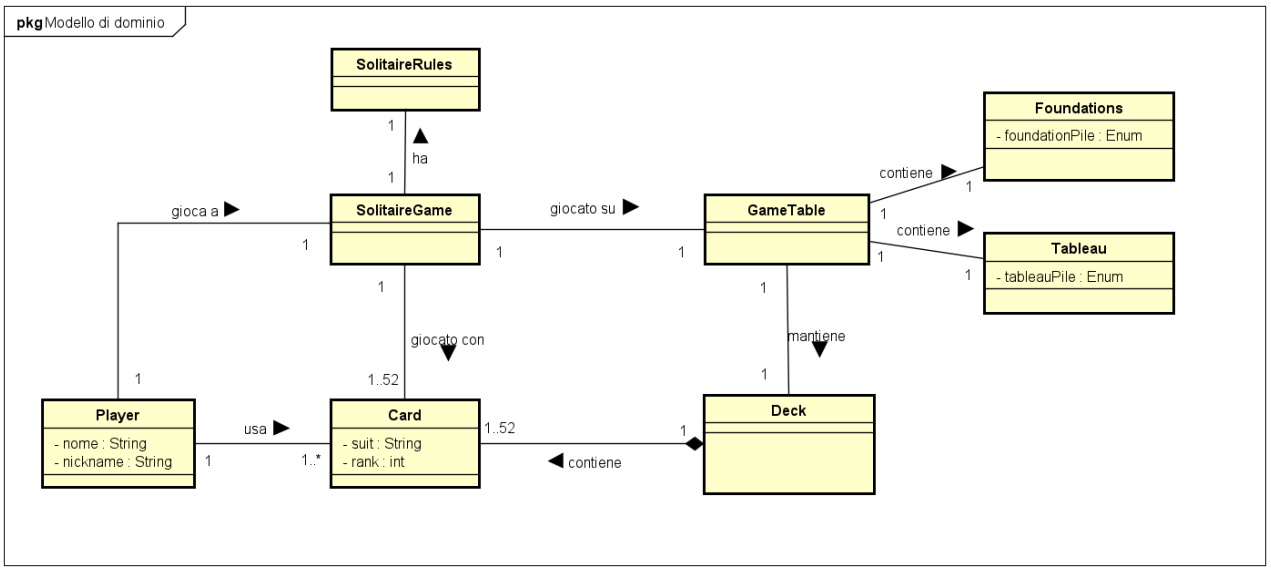
GameTable: Rappresenta il luogo in cui viene svolto il gioco

Foundations: Rappresenta la zona del tavolo da gioco in cui sono presenti quattro pile di carte inizialmente vuote (basi)

Tableau: Rappresenta la zona del tavolo da gioco in cui sono presenti sette colonne di carte. Su queste verranno distribuite le carte in modo che ogni colonna ne abbia un numero uguale alla sua posizione (la prima colonna una, la seconda due, ecc..), per un totale di 28 carte presenti nel Tableau all'inizio del gioco

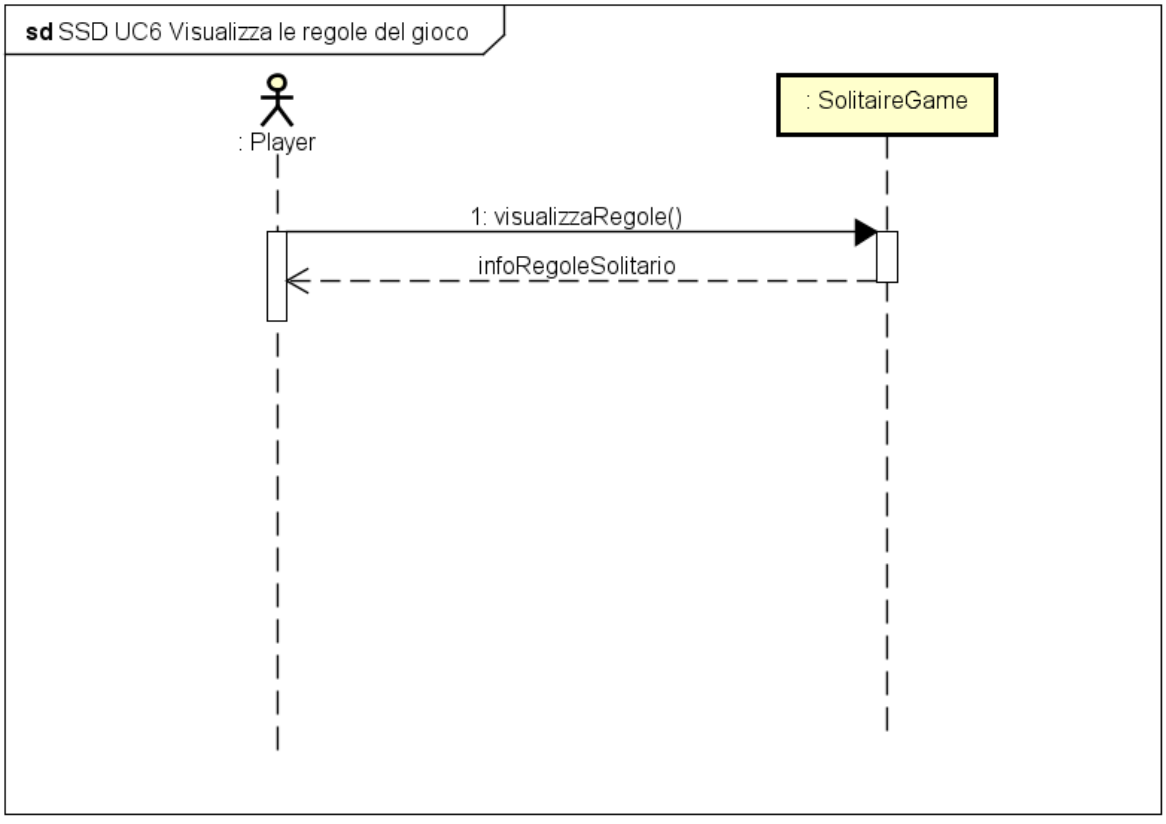
SolitaireRules: rappresenta la vista che viene mostrata quando l'utente vuole leggere le regole del gioco

Tenendo conto di associazioni e attributi tra queste classi, il modello di dominio che ne viene fuori è il seguente:

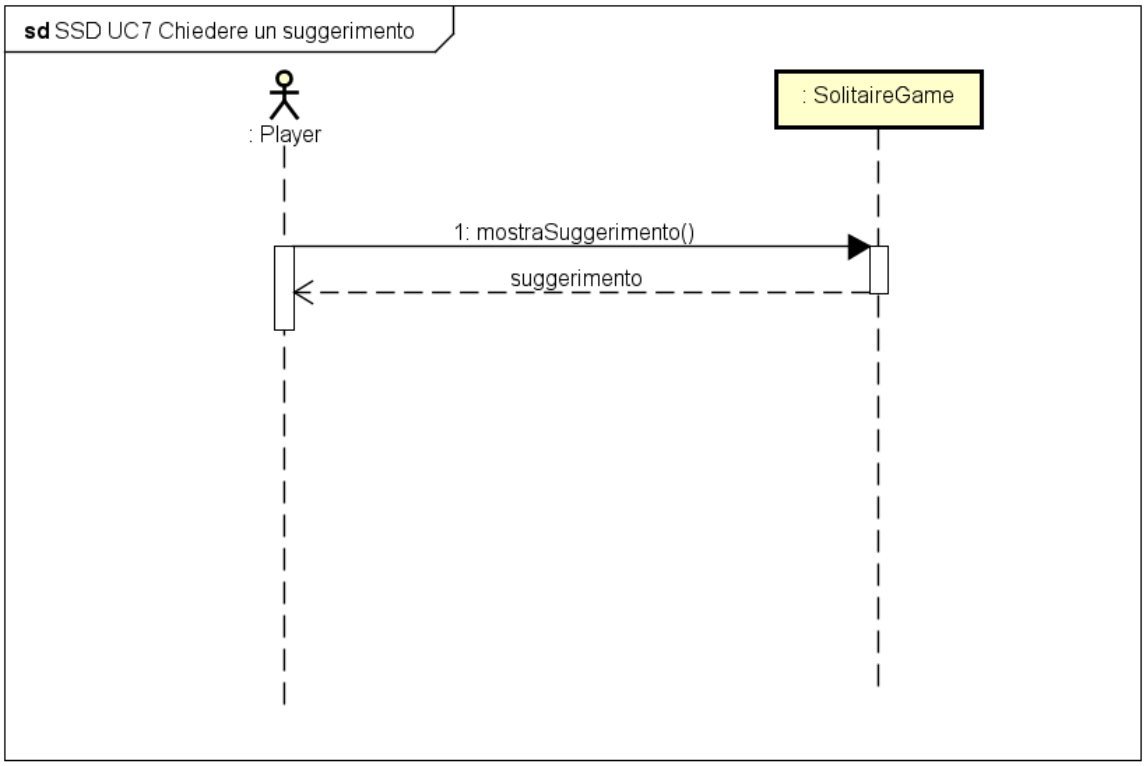


3.2 Diagrammi di sequenza di sistema

Procediamo ora con il secondo step dell'analisi orientata agli oggetti, con la creazione del diagramma di sequenza di sistema (SSD), al fine di illustrare il corso degli eventi di input e output costituenti il caso d'uso in analisi UC6, e nello specifico come già detto lo scenario principale di successo. Avremo allora:



Per quanto riguarda invece il caso d'uso UC7, il diagramma di sequenza di sistema sarà il seguente:



3.3 Contratti delle operazioni

Il prossimo passo è quello della descrizione delle operazioni individuate all'interno del SSD tramite i contratti delle operazioni. Riguardo ai casi d'uso UC6 e UC7, le operazioni di sistema che vengono analizzate sono:

Contratto CO1: Visualizza Regole

Operazione: visualizzaRegole()

Riferimenti: Caso d'uso: Visualizza le regole del gioco

Pre-condizioni:

- è in corso una partita di SolitaireGame

Post-condizioni:

- è stata creata un'istanza *sr* di SolitaireRules;
- gli attributi di *sr* sono stati inizializzati;
- l'istanza *sr* è stata associata a SolitaireGame tramite l'associazione "*ha*";

Contratto CO2: Mostra Suggerimento

Operazione: mostraSuggerimento()

Riferimenti: Caso d'uso: Chiedere un suggerimento

Pre-condizioni:

- è in corso una partita di SolitaireGame

Post-condizioni:

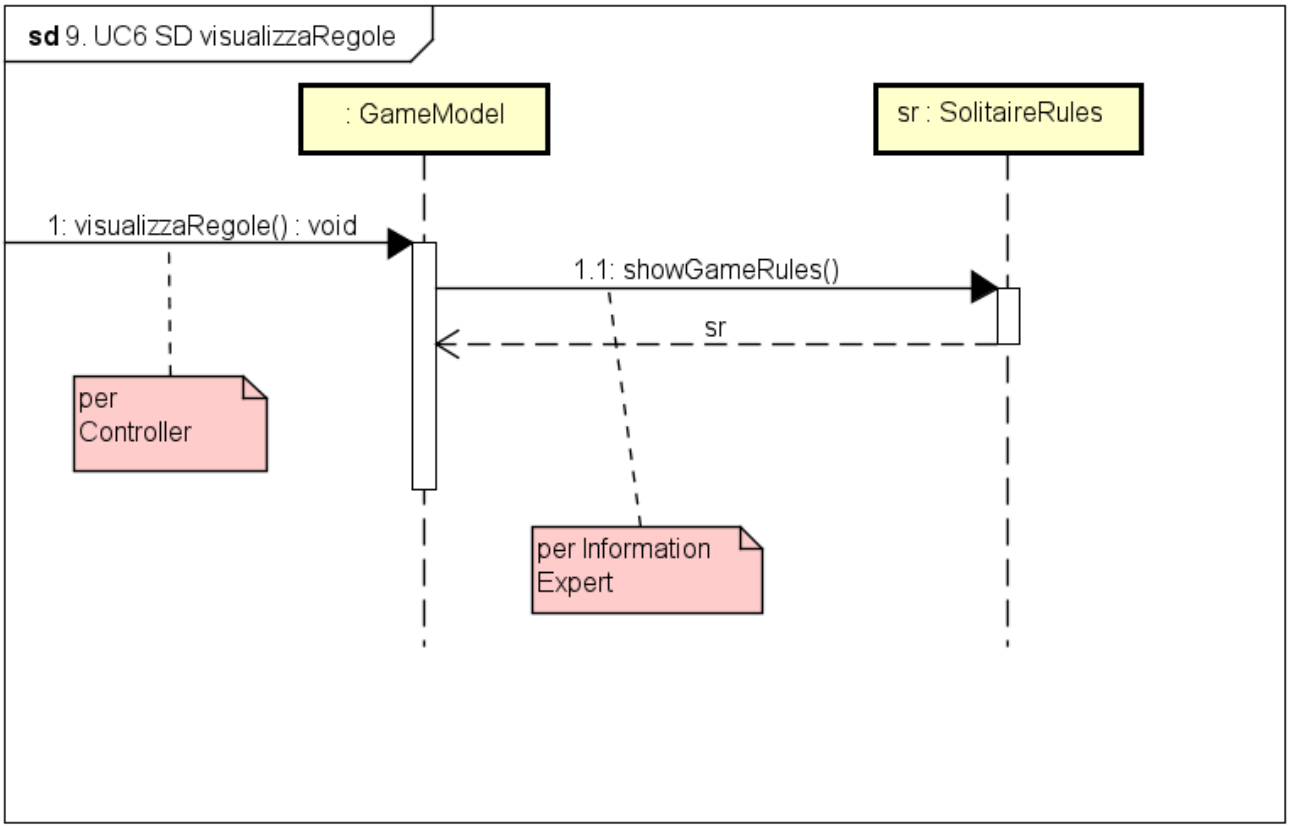
- il giocatore avrà visualizzato le carte che è possibile spostare per continuare il gioco;

4 Progettazione

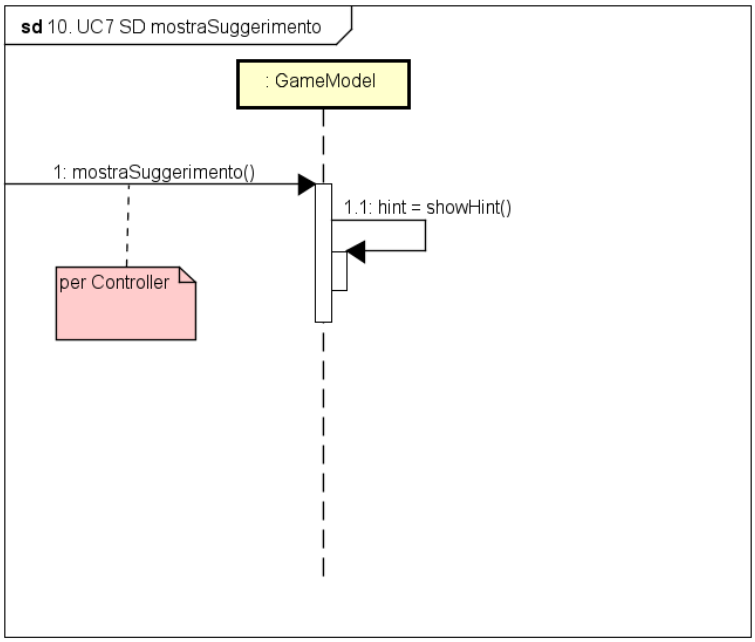
L'elaborato principale di questa fase che è stato preso in considerazione è il **Modello di Progetto**, ovvero l'insieme dei diagrammi che descrivono la progettazione logica sia da un punto di vista dinamico (Diagrammi di Interazione) che da un punto di vista statico (Diagramma delle Classi). Seguono dunque i diagrammi di Interazione più significativi e il diagramma delle Classi relativi ai casi d'uso UC6 e UC7, determinati a seguito di un attento studio degli elaborati scritti in precedenza.

4.1 Diagrammi di sequenza

- **Visualizza regole**

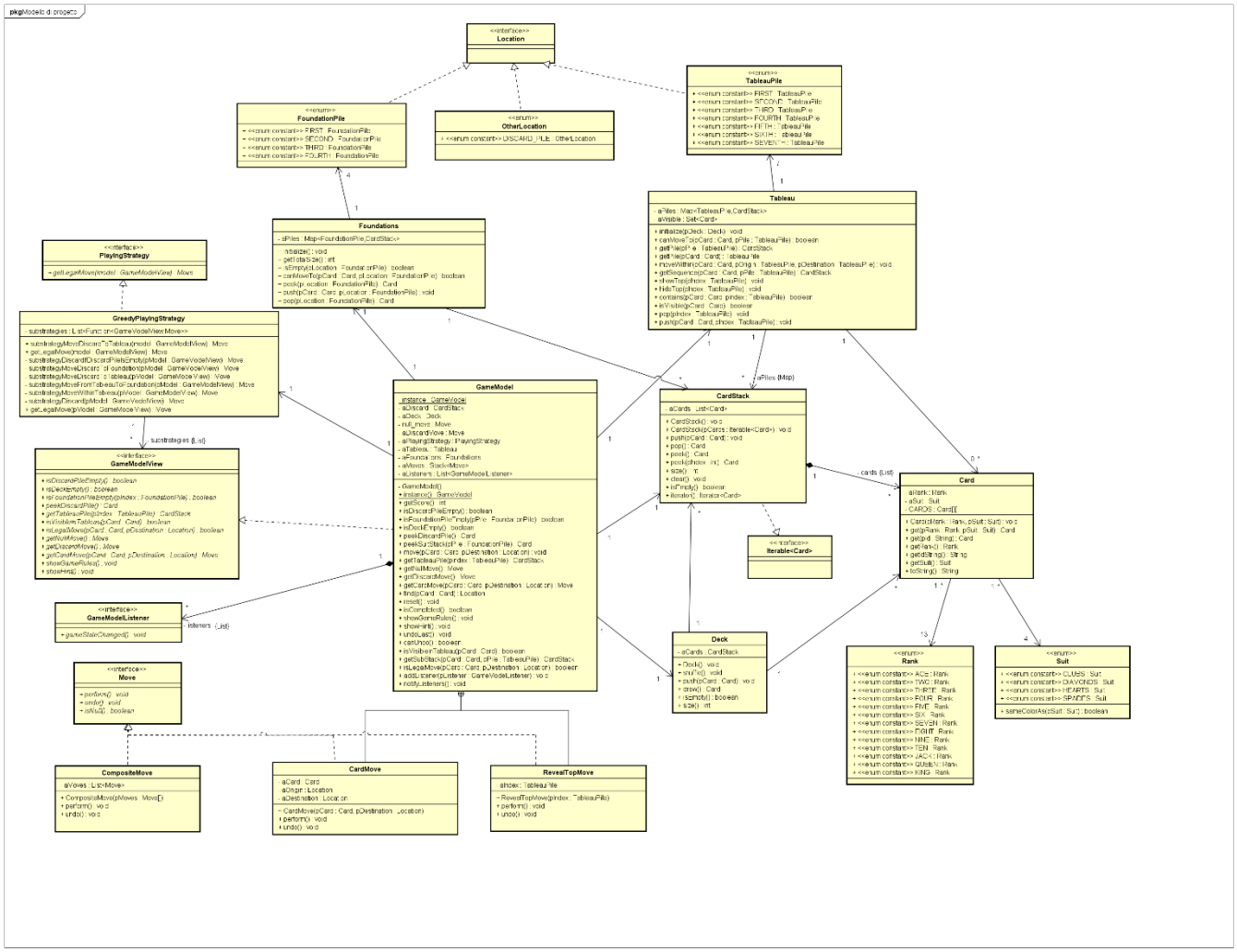


- **Mostra suggerimento**



4.2 Diagramma delle classi

Al fine di migliorare la leggibilità, il Diagramma delle Classi è riportato anche nella cartella immagini presente in questa iterazione. Il nome dell'immagine è "*DCD.png*".



5 Testing

Si è scelto di implementare test a scatola nera, testando le singole classi implementate (test unitari) ed in particolare ogni loro metodo, seguendo un approccio bottom-up, ovvero collaudando dapprima le unità più interne del sistema, per poi passare alle classi che dipendono da quest'ultime. È stato utilizzato un approccio bottom-up, in modo tale da evitare di testare metodi (in genere di classi che si trovano in strati più esterni del sistema) il cui unico compito rilevante è la chiamata di metodi già testati. Questo, inoltre, ha permesso di ridurre in parte il numero di test effettuati.

5.1 Casi di test

I test sono stati eseguiti nell'ordine, seguendo un approccio bottom-up, sulle classi *Deck, CardStack, Tableau, Foundations* e *GameModel*:

- **Deck**
  - Shuffle
    - Vengono creati due mazzi mischiati in modo random e si verifica che i due oggetti non sono uguali
  - Draw
    - Vengono pescate un certo numero di carte dal mazzo e si verifica che la nuova dimensione di quest'ultimo sia quella corretta
  - Size
    - Si verifica che inizialmente il mazzo creato è composto da 52 carte
    - Si pesca una carta e si verifica che la nuova dimensione sia quella corretta

- **CardStack**
  - Push
    - Si inserisce una carta in uno stack inizialmente vuoto e si verifica che la nuova dimensione dello stack è effettivamente 1
  - Pop
    - Si rimuove una carta da uno stack e si verifica che la nuova dimensione dello stack è effettivamente quella originale decrementata di 1
  - Peek
    - Si inserisce una carta in cima allo stack e si verifica che la carta selezionata con il metodo peek sia effettivamente quella inserita
  - Size
    - Si inseriscono un certo numero di carte nello stack con il metodo push e si verifica che la nuova dimensione dello stack sia quella corretta
  - Clear
    - Si inseriscono inizialmente un certo numero di carte in uno stack vuoto con il metodo push e si verifica che, in seguito all'esecuzione del metodo clear, la nuova dimensione dello stack sia zero
    - Si crea uno stack composto da 52 carte mischiate e si verifica che, in seguito all'esecuzione del metodo clear, la nuova dimensione dello stack sia zero

- isEmpty
  - Si verifica che inizialmente lo stack creato è vuoto (dimensione = 0)
  - Si inseriscono un certo numero di carte e si verifica che lo stack non è vuoto (dimensione ≠ 0)

- **Tableau**
  - getTableauPile
    - Si verifica che, ad esempio, nella seconda pila di carte del Tableau siano presenti effettivamente due carte
  - Contains
    - Si verifica se una determinata carta è presente o meno nel Tableau
  - canMoveTo
    - Si verifica se una determinata carta può essere spostata in cima a una specifica pila di carte del Tableau, in base alle regole del gioco
  - getSequence
    - Si verifica che la sequenza di carte restituita abbia una dimensione effettivamente minore rispetto a quella dello stack in cui si trova inizialmente la carta selezionata
  - Pop
    - Si rimuove una carta da una specifica pila di carte del Tableau e si verifica che la nuova dimensione di questa sia quella corretta
  - Push
    - Si inserisce una carta in una specifica pila di carte del Tableau e si verifica che la nuova dimensione di questa sia quella corretta

- **Foundations**
  - isEmpty
    - Si verifica che inizialmente le quattro pile di carte presenti in Foundations siano vuote
  - Peek
    - Si inserisce una carta in una delle quattro pile di carte di Foundations e si verifica che la carta selezionata con il metodo peek sia effettivamente quella inserita
  - Push
    - Si inserisce una carta in una specifica pila di carte di Foundations e si verifica che la nuova dimensione di questa sia quella corretta

- Pop
  - Si rimuove una carta da una specifica pila di carte di Foundations e si verifica che la nuova dimensione di questa sia quella corretta

- canMoveTo
  - Si verifica se una determinata carta può essere spostata in cima a una specifica pila di carte di Foundations, in base alle regole del gioco

- **GameModel**
  - Find
    - Si verifica la posizione di una carta in una determinata zona del tavolo da gioco. Sono state analizzate le tre diverse posizioni in cui una carta può trovarsi e da cui può essere spostata:
      - Caso 1: la carta si trova nella pila delle carte scartate dal mazzo (Discard Pile)
      - Caso 2: la carta si trova in una delle sette pile di carte del Tableau
      - Caso 3: la carta si trova in una delle quattro pile di carte di Foundations