02 – Elaborazione - Iterazione 2

1 Introduzione

Durante questa seconda iterazione ci si concentrerà sui seguenti requisiti:

- Di nuovo, implementare uno scenario base del caso d'uso UC1: *Gioca partita Gioco dell'Oca:* i giocatori, con le proprie pedine, si spostano lungo le caselle del tabellone. Come nell'iterazione precedente, avviare il gioco come una simulazione non richiede nessun input dall'utente, se non quello di inserire il numero di giocatori. Tuttavia, nell'iterazione 2, vengono applicate alcune istruzioni associate a delle caselle particolari. Queste verranno descritte nei punti successivi

- Quando un giocatore arriva su una *CasellaAvanti*, questo si sposterà in avanti di un certo numero X di caselle, con X numero intero positivo

- Quando un giocatore arriva su una *CasellaIndietro*, questo si sposterà indietro di un certo numero Y di caselle, con Y numero intero positivo

- Il primo giocatore che arriva (o supera) sulla *CasellaArrivo* avrà vinto la partita

- La *CasellaPartenza* (di indice 0) e la *CasellaArrivo* (di indice 62) non possono avere istruzioni associate

- Implementazione dei casi di test, utilizzando il framework jUnit, che permettono di rilevare eventuali malfunzionamenti del codice.

2 Presentazione del caso d'uso UC1

Andiamo nuovamente a presentare lo schema del caso d'uso *UC1: Gioca partita Gioco dell'Oca*.

UC1: Gioca partita Gioco dell'Oca

Nome del caso d'uso	**UC1: *Gioca partita Gioco dell'Oca***
Portata	Applicazione Gioco dell'Oca
Livello	obiettivo utente
Attore primario	Utente del Sistema
Parti interessate e Interessi	▪ **Utente:** vuole fare una simulazione del Gioco dell'Oca, nel modo più fluido possibile. Vuole una visualizzazione chiara degli elementi presenti sul tabellone di gioco.
Precondizioni	L'utente ha avviato la simulazione di una partita del Gioco dell'Oca.

Garanzia di successo (post-condizioni)	Viene mostrata a schermo la simulazione della partita, con i vari round e i turni eseguiti dai giocatori in ogni round. Per ogni round, inoltre, l'utente ha la possibilità di vedere su quale casella si trova ogni giocatore e il valore del dado lanciato che indica di quante caselle si dovrà muovere.
Scenario principale di successo	1. L'Utente richiede l'inizializzazione di una nuova partita e inserisce il numero di giocatori. 2. L'Utente inizia a giocare. 3. Il Sistema visualizza la traccia del gioco per la successiva mossa del giocatore. Il passo 3 viene ripetuto fino a quando non c'è un vincitore o fino a quando l'Utente non termina di giocare.
Estensioni (o flussi alternativi)	*a. In qualsiasi momento, l'applicazione termina in modo anomalo: per consentire il ripristino, bisogna garantire che il sistema possa essere ripristinato, a partire da qualsiasi passo dello scenario. 1. L'utente riavvia l'applicazione e richiede il ripristino dello stato precedente. 2. L'applicazione ricostruisce lo stato precedente. 2a. L'applicazione rileva delle anomalie che impediscono il ripristino: 1. L'applicazione segnala un errore all'utente, registra l'errore, e passa in uno stato pulito. 2. L'utente inizia una nuova partita.
Requisiti speciali	
Elenco delle varianti tecnologiche e dei dati	
Frequenza di ripetizione	potrebbe essere quasi ininterrotta.
Varie	

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

Le nuove classi concettuali CasellaRegolare, CasellaAvanti, CasellaIndietro, CasellaArrivo che emergono in questa seconda iterazione sono tutte simili tra loro, sono variazioni di una Casella. In questa situazione, è possibile organizzarle in una gerarchia di classi di generalizzazione-specializzazione in cui la superclasse Casella rappresenta un concetto più generale e le sottoclassi dei concetti più specializzati. In UML, le relazioni di generalizzazione-specializzazione sono rappresentate con una freccia triangolare che punta dalle classi specializzate alla classe più generale.

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

GiocoDellOca: Rappresenta l'applicazione del gioco *Gioco dell'Oca*

Tabellone: Rappresenta il luogo in cui viene svolto il gioco

Casella: Rappresenta una generica casella nel tabellone di gioco, caratterizzata da un nome e da un numero intero, ovvero il suo indice. In totale sono presenti 63 caselle sul tabellone

CasellaRegolare: Rappresenta una casella a cui non è associata nessuna istruzione particolare

CasellaAvanti: Rappresenta una casella che ha associata l'istruzione di avanzare di un certo numero X di caselle

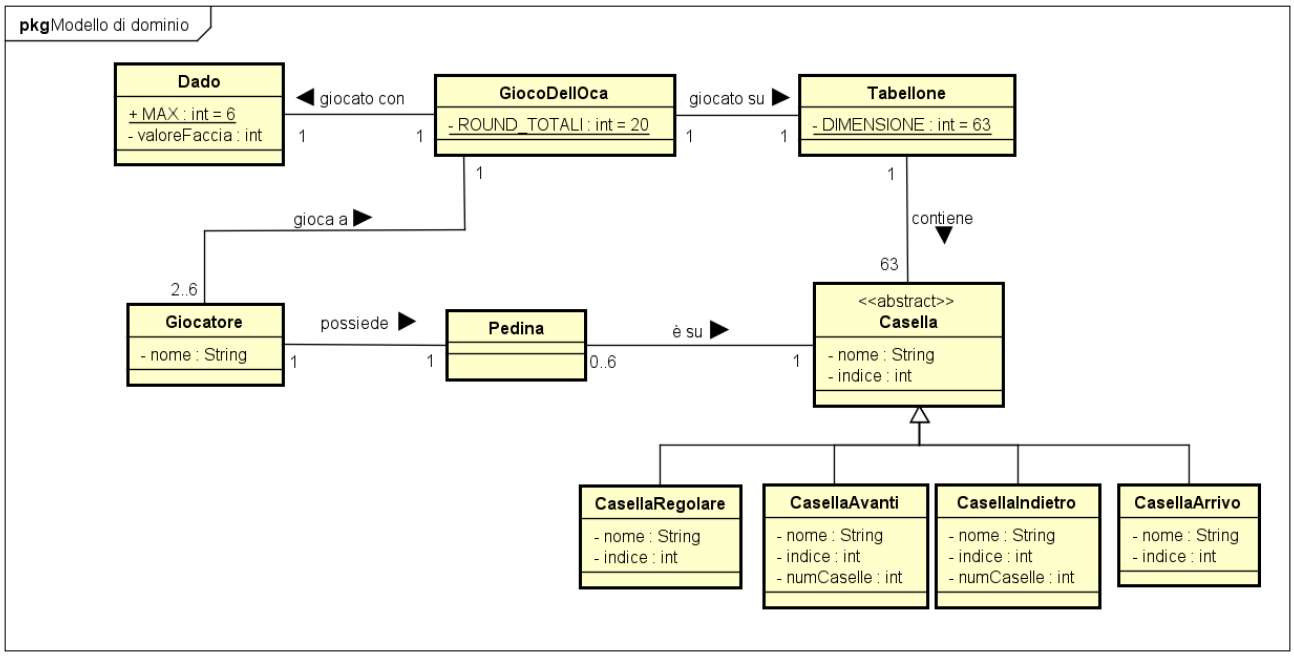
CasellaIndietro: Rappresenta una casella che ha associata l'istruzione di andare indietro di un certo numero Y di caselle

CasellaArrivo: Rappresenta l'ultima casella, la numero 63. Il primo giocatore che arriva su di essa (o la supera) vince la partita

Dado: Rappresenta un dado con 6 facce, utilizzato in ogni turno da ciascun giocatore. Il suo valore dopo un lancio indica di quante caselle dovrà avanzare il giocatore con la propria pedina

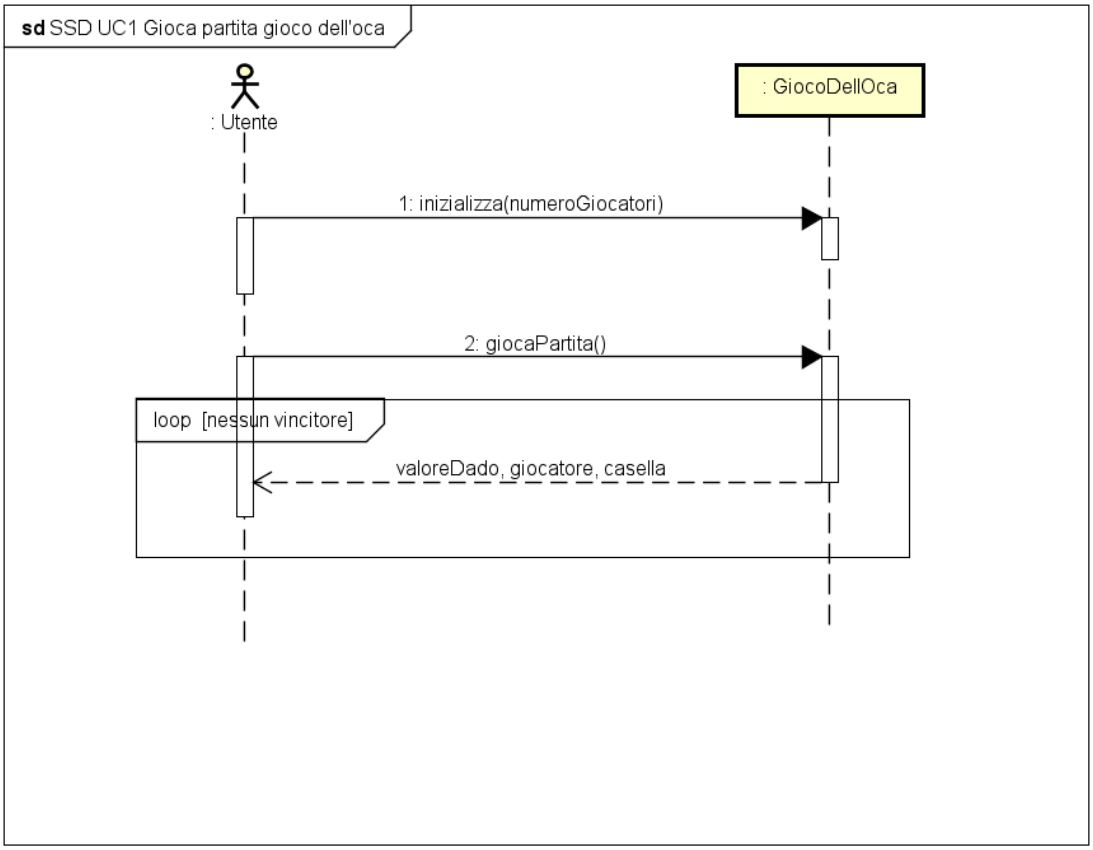
Pedina: Rappresenta l'oggetto utilizzato da ogni giocatore per spostarsi tra le varie caselle presenti nel tabellone di gioco. Ciascun giocatore ne può avere una soltanto

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



3.2 Diagramma 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 UC1, e nello specifico come già detto lo scenario principale di successo. Da questa iterazione, non emergono nuovi diagrammi di sequenza di sistema. Viene quindi riportato il diagramma di sequenza dell'iterazione precedente.



3.3 Contratti delle operazioni

Per questa iterazione non ci sono nuovi contratti delle operazioni da analizzare.

4 Progettazione

La progettazione orientata agli oggetti è la disciplina di UP interessata alla definizione degli oggetti software, delle loro responsabilità e a come questi collaborano per soddisfare i requisiti individuati nei passi precedenti. 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 all'iterazione 2, determinati a seguito di un attento studio degli elaborati scritti in precedenza.

In questa seconda iterazione, si è deciso di creare un'operazione polimorfica per ogni tipo per il quale il comportamento varia. Esso varia per le classi CasellaAvanti, CasellaIndietro e CasellaArrivo. Dato che ciò che varia è il comportamento del giocatore quando arriva su una determinata casella, è stato scelto *arrivatoSu*() come nome per l'operazione polimorfica.

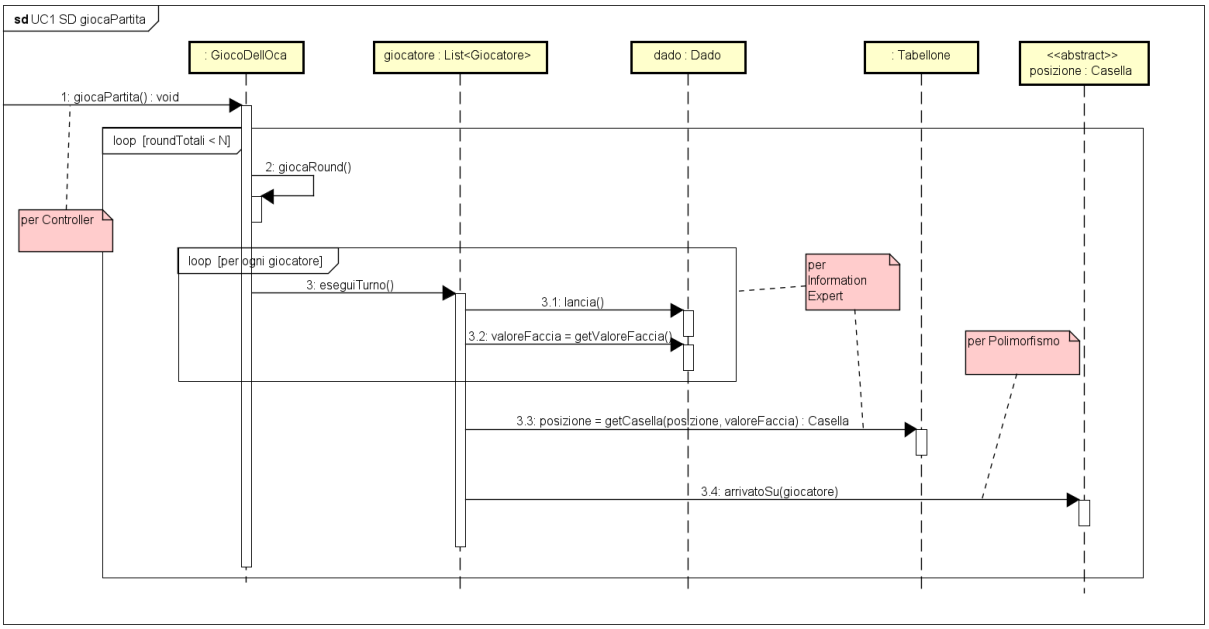
Per il pattern GRASP Polimorfismo, si creano quindi tante classi separate quanti sono i diversi comportamenti associati all'operazione *arrivatoSu*(); si implementa poi il metodo in ciascuna di queste sottoclassi.

Si è scelto, inoltre, di utilizzare il design pattern creazionale Singleton per le classi *GiocoDellOca* e *Tabellone*, affinché queste abbiano un'unica istanza e sia possibile fornire un punto di accesso globale all'unica istanza. Per fare questo, si è definito per ciascuna delle due classi un singolo costruttore privato e un metodo pubblico che crea (se non è presente) o restituisce un riferimento all'unica istanza.

I diagrammi di sequenza risultanti, in seguito all'applicazione di questo pattern, sono i seguenti:

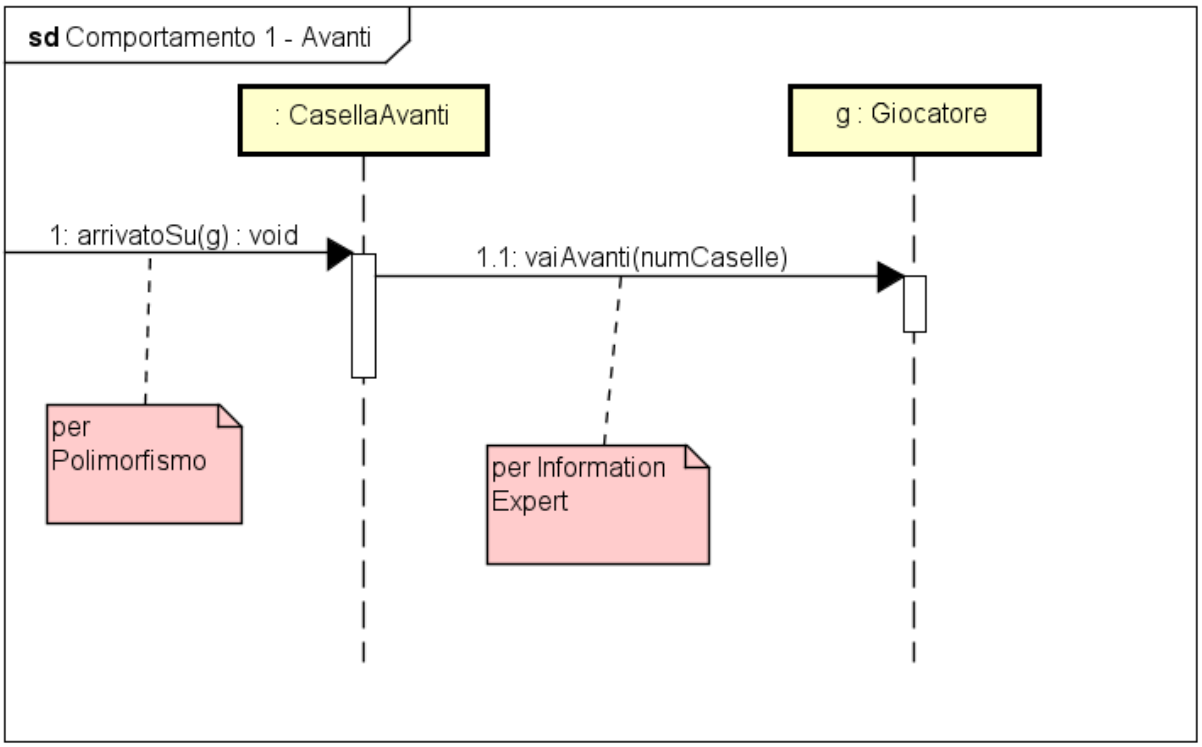
4.1 Diagrammi di sequenza

- **Gioca partita**

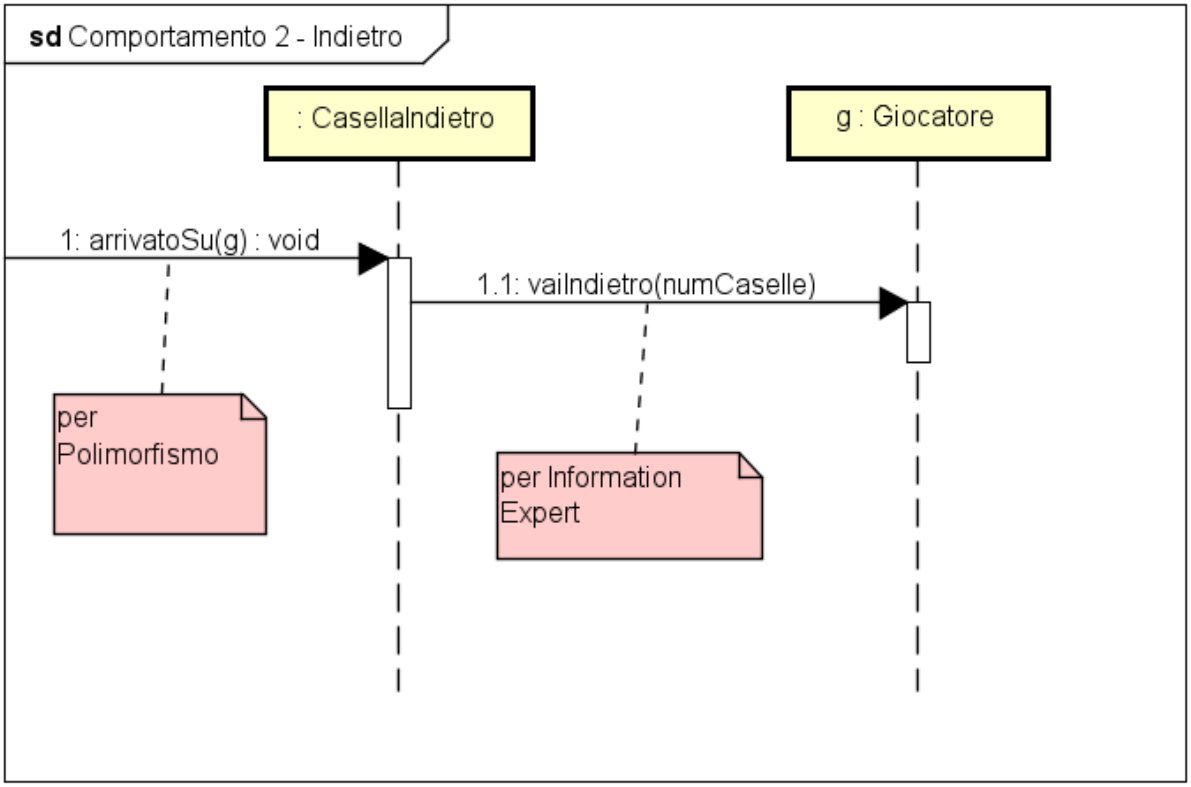


I diversi comportamenti polimorfici vengono mostrati qui sotto in diagrammi di sequenza separati; in questo modo si ha un diagramma per ogni alternativa possibile.

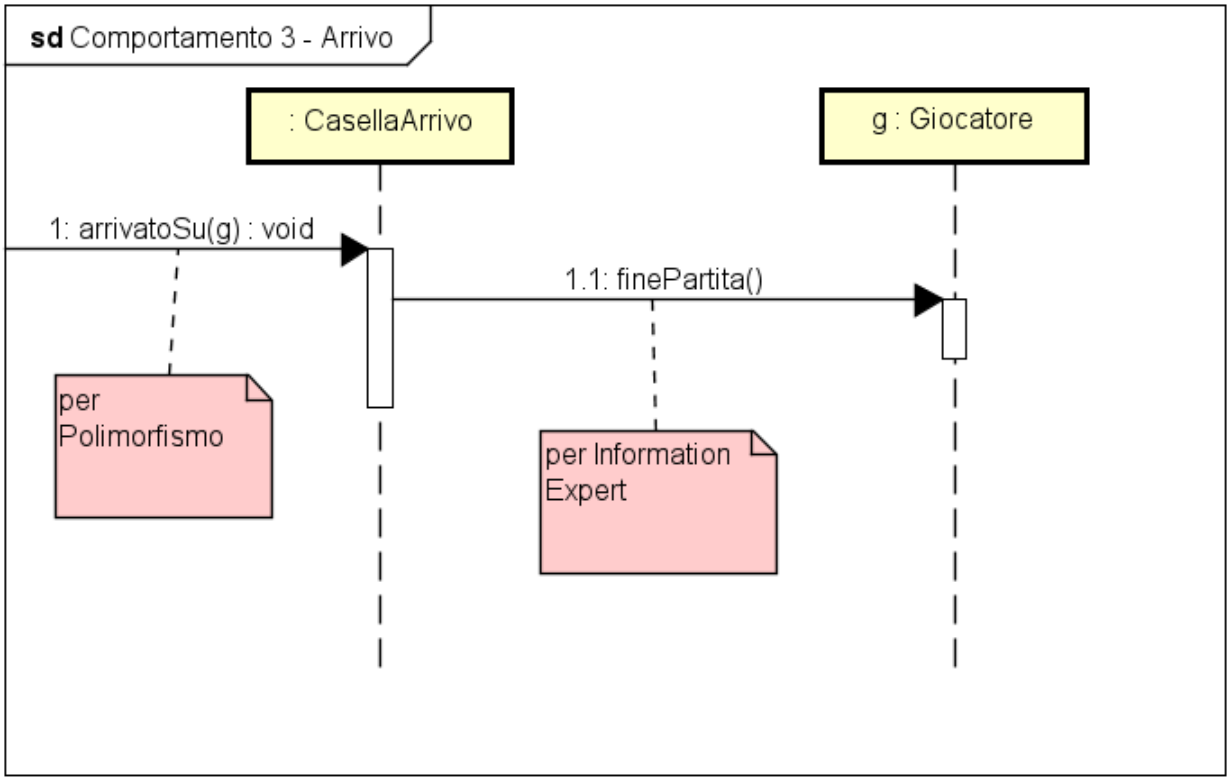
- **Comportamento 1 – Casella Avanti**



- **Comportamento 2 – Casella Indietro**



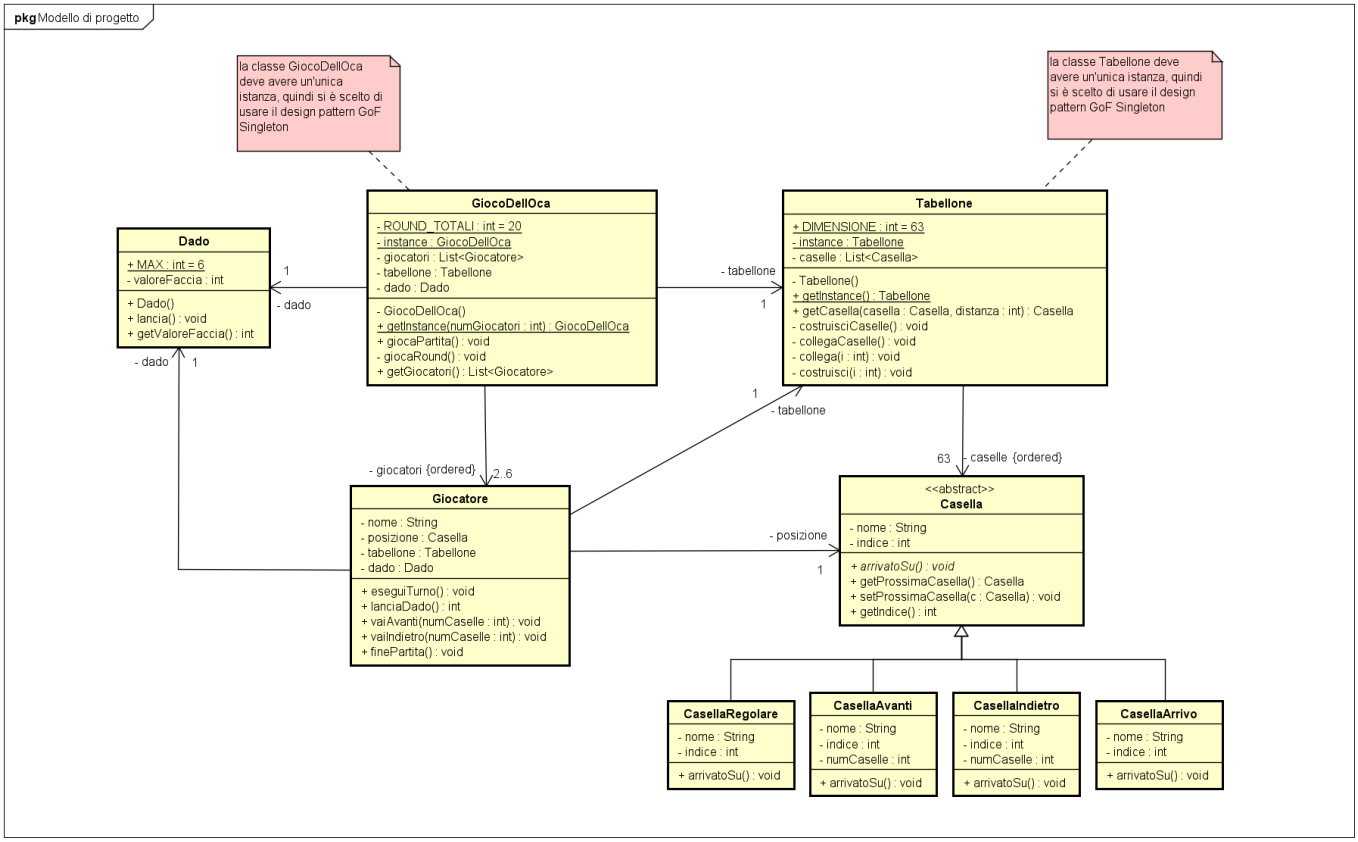
- **Comportamento 3 – Casella Arrivo**



4.2 Diagramma delle classi

Nel diagramma delle classi qui sotto riportato, si può vedere come in questa iterazione sia stata ridefinita la progettazione, in modo che adesso sia il *Giocatore* a conoscere la casella in cui si trova e non più la sua *Pedina*. Così facendo, si migliora l'accoppiamento tra i due oggetti *Giocatore* e *Casella*.

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 *Dado, Tabellone, Giocatore* e *GiocoDellOca*.

- **Dado**
  - Lancia
    - Si verifica che il lancio di un dado restituisce sempre un numero intero casuale compreso tra 1 e 6, cioè il valore di una delle sei facce del dado

- **Tabellone**
  - getCasella
    - Si verifica che, partendo da una casella di indice X, il metodo restituisce una casella di indice X + Y, dove Y rappresenta per esempio il valore dopo il lancio del dado che indica di quante caselle bisogna spostarsi in avanti
  - getCasellaPartenza
    - Si verifica che il metodo restituisce la prima casella del tabellone di gioco, ovvero quella con indice 0
  - getCasellaArrivo
    - Si verifica che il metodo restituisce l'ultima casella del tabellone di gioco, ovvero quella con indice 62

- **Giocatore**
  - eseguiTurno
    - Si verifica che, in seguito a un turno fatto da un giocatore, la posizione di questo sia diversa da quella da cui egli è partito. In base alla casella di arrivo, inoltre, si applica l'eventuale istruzione associata.
  - lanciaDado
    - Si verifica che il valore restituito in seguito al lancio di un dado sia un numero casuale compreso tra 1 e 6
  - vaiAvanti
    - Si verifica che all'arrivo su una *CasellaAvanti*, questo metodo faccia avanzare il giocatore di un numero di caselle compreso tra 1 e 3
  - vaiIndietro
    - Si verifica che all'arrivo su una *CasellaIndietro*, questo metodo faccia tornare indietro il giocatore di un numero di caselle compreso tra 1 e 3
  - getPosizione
    - Si verifica che il metodo restituisce la casella attuale in cui si trova il giocatore

- **GiocoDellOca**
  - getGiocatori
    - Si verifica che all'inizio della simulazione della partita tutti i giocatori si trovano nella casella di partenza con indice 0