DigiGoose! Il gioco dell’oca digitale

Sommario

1 - Ideazione e analisi dei requisiti 5
1.1 Introduzione 5
1.2 Requisiti 5
1.3 Obiettivi e casi d'uso 5
1.4 Modello dei casi d'uso 6
1.4.1 UC1: Avviare una nuova partita 6
1.4.2 UC2: Tirare i dadi 7
1.4.3 UC3: Muovere la pedina 8
1.4.4 UC4: Gestire le caselle speciali 8
1.4.5 Gestire i turni 9
1.4.6 UC6: Determinare il vincitore 9
1.5 Documento di visione 9
1.5.1 Introduzione 9
1.5.1.1 Scopo 9
1.5.1.2 Portata 9
1.5.1.3 Definizioni, acronimi e abbreviazioni 9
1.5.1.4 Riferimenti 9
1.5.2 Posizionamento 9
1.5.2.1 Opportunità di business 9
1.5.2.2 Formulazione del problema 9
1.5.2.3 Formulazione della posizione del prodotto 10
1.5.3 Parti interessate e descrizioni utente 10
1.5.3.1 Riepilogo delle parti interessate 10
1.5.3.2 Riepilogo dell'utente 10
1.5.3.3 Ambiente dell'utente 10
1.5.3.4 Utente (Giocatore) 10
1.5.3.5 Sviluppatore 11
1.5.3.6 Alternative e concorrenza 11
1.5.4 Descrizione generale del prodotto 11
1.5.4.1 Punto di vista del prodotto 11
1.5.4.2 Riepilogo dei vantaggi 11
1.6 Regole di business 12
1.7 Specifiche supplementari 13
1.8 Glossario 14
2 – Elaborazione – Iterazione 1 15

2.1 Introduzione 15
2.2 Aggiornamento caso d'uso UC1 15
2.3 Analisi orientata agli oggetti 16
2.2.1 Modello di dominio 16
2.2.2 Diagramma di sequenza di sistema 17
2.2.3 Contratti delle operazioni 18
2.4 Progettazione 24
2.4.1 Diagrammi di sequenza 24
2.4.2 Diagramma delle classi 27
3 – Elaborazione – Iterazione 2 28
3.1 Introduzione 28
3.2 Casi d'uso in dettaglio 28
3.2.1 UC1: Avviare una nuova partita 28
3.2.2 UC2: Tirare i dadi 29
3.2.3 UC3: Muovere la pedina 31
3.2.4 UC4: Gestire le caselle speciali 32
3.2.5 UC5: Gestire i turni 33
3.2.6 UC6: Determinare il vincitore 35
3.3 Changelog completo dell'elaborazione 36
3.3.1 Modifiche principali 36
3.3.1.1 Configurazione partita 36
3.3.1.2 Gestione giri e turni 36
3.3.1.3 Meccanica di sblocco giocatore 36
3.3.1.4 Movimento ed effetti caselle 36
3.3.1.5 Interfaccia utente 37
3.4 Contratti delle operazioni 37
3.5 Progettazione aggiornata 42
3.5.1 Diagramma di sequenza 42
4 – Elaborazione – Iterazione 3 51
4.1 Introduzione 51
4.2 Implementazione dell'interfaccia utente con Swing 51
4.2.1 Configurazione della partita 51
4.2.1.1 Selezione del numero di giocatori e configurazione individuale 51
4.2.1.2 Raccolta, validazione e conferma delle impostazioni 52
4.2.2 Tabellone di gioco 52
4.2.2.1 Struttura e layout del tabellone 52

4.2.2.2 Rendering delle caselle e delle informazioni 52
4.2.2.3 Visualizzazione delle pedine dei giocatori 53
4.2.2.4 Aggiornamento del board durante la partita 53
4.2.3 Visualizzazione effetti caselle speciali 53
4.3 Sistema di salvataggio e caricamento partita con JSON 53
4.3.1 Creazione del file JSON per lo stato del gioco 54
4.3.2 Gestione del caricamento della partita 54
4.4 Irrobustimento della logica di gioco e correzioni bug 55
4.4.1 Gestione bug rilancio dei dadi 55
4.4.2 Gestione turni giocatori bloccati 56

1 - Ideazione e analisi dei requisiti

1.1 Introduzione

L'obiettivo della fase di ideazione è di condurre un'analisi preliminare per delineare un'idea di massima del gioco da sviluppare, ottenendo stime di fattibilità che considerino tempi e risorse necessari.

Per definire e comprendere al meglio i requisiti del gioco, durante la fase di ideazione verranno esaminate le seguenti sezioni di questo capitolo.

1.2 Requisiti

DigiGoose! ha lo scopo di creare una buona esperienza di gioco del gioco dell'oca moderno in versione digitale, seguendo le regole della versione moderna del gioco dell'oca Clementoni. L'applicazione deve rappresentare uno strumento interattivo e coinvolgente che guidi il giocatore attraverso le fasi essenziali del gioco. In particolare:

- L'applicazione si concentrerà sulla creazione della struttura fondamentale del gioco, implementando le funzionalità di base come il movimento delle pedine, la gestione delle caselle speciali del tabellone e il lancio dei dadi.
- L'applicazione permetterà ai giocatori di visualizzare un tabellone predefinito, scegliere il colore della propria pedina, inserire i nomi dei giocatori e scegliere se il giocatore è umano o gestito dal computer.
- L'applicazione includerà la funzionalità di salvare la partita corrente e di caricarla in un secondo momento, tramite l'opzione "Riprendi Partita".

All'avvio di una nuova partita, i giocatori inseriranno i nomi dei partecipanti e selezioneranno il colore delle pedine. Il sistema inizializzerà il gioco con il tabellone e le pedine posizionate. Durante il gioco, l'applicazione aggiornerà la sessione corrente, registrando ogni mossa ed evento.

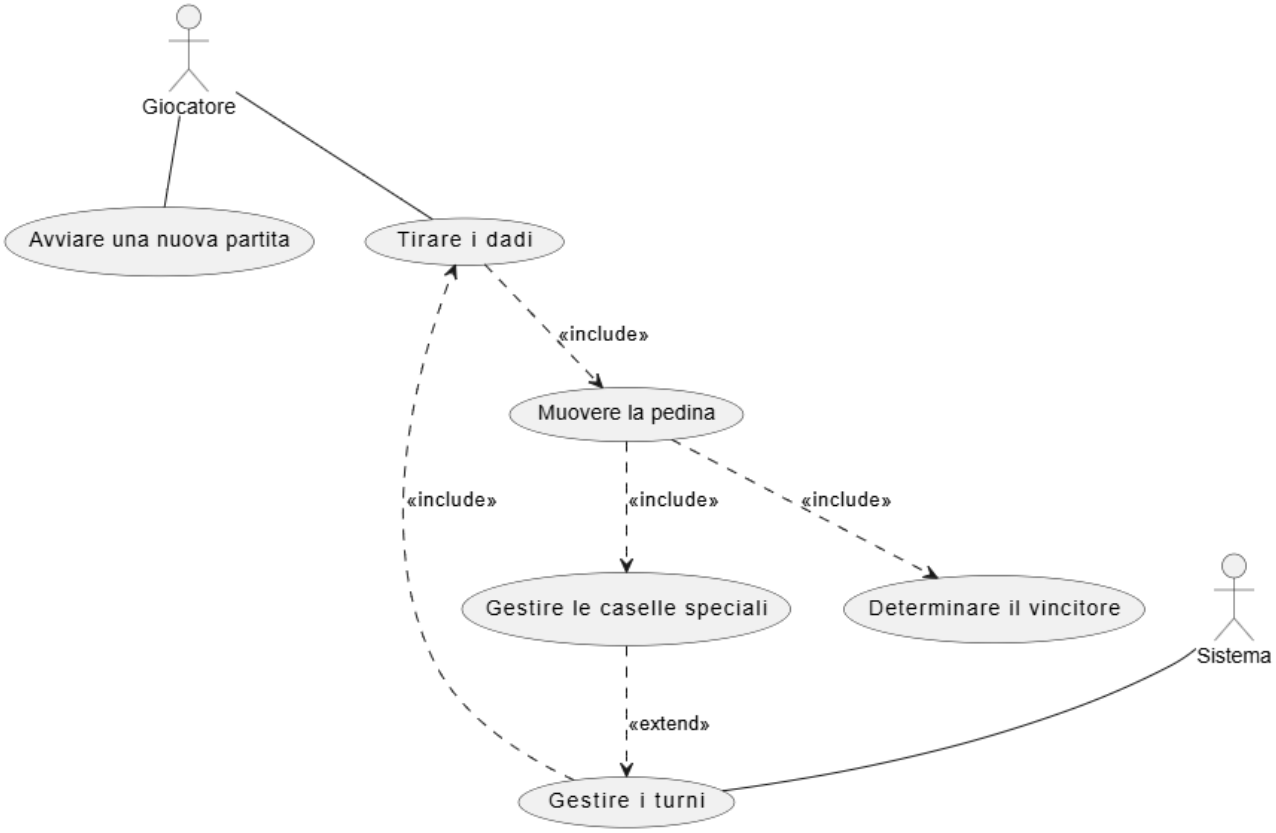
1.3 Obiettivi e casi d'uso

Analizzando i requisiti, al fine di iniziare lo sviluppo del gioco, sono stati identificati l'attore principale a cui è destinato il sistema e gli obiettivi che egli intende portare a termine per quanto riguarda le funzionalità base; sono, quindi, stati ricavati i casi d'uso principali, che sono 6:

Attore	Obiettivo	Caso d'uso
Giocatore	All'apertura dell'applicazione, inizia una nuova partita	UC1: Avviare una nuova partita
Giocatore	A inizio turno, vengono tirati i dadi per determinare il movimento.	UC2: Tirare i dadi
Sistema	In base ai dadi, viene mossa la pedina sul tabellone.	UC3: Muovere la pedina
Sistema	In base a dove si ferma la pedina, viene gestita la casella, se speciale.	UC4: Gestire le caselle speciali
Sistema	Una volta finito il movimento e gestita la casella, viene gestito il prossimo a tirare.	UC5: Gestire i turni

Sistema	Se in UC3 si finisce sulla casella 63, viene determinato il vincitore	UC6: Determinare il vincitore

C'è da notare che "Sistema" non è un attore, anche se viene rappresentato come tale, poiché non rappresenta un'entità esterna che interagisce con il sistema stesso. Tuttavia, per rendere più chiaro che i casi d'uso da UC3 a UC6 non coinvolgono direttamente l'attore "Giocatore", nella tabella è stato collocato nella stessa colonna degli attori. Inoltre, nelle prossime iterazioni, con l'introduzione del controllo della CPU per il giocatore, questo elemento, pur essendo interno al sistema, ne simulerà il comportamento, che invece appartiene a un'entità esterna. Infine, il sistema può essere visto come un "arbitro" che verifica il corretto svolgimento delle operazioni, dando l'impressione di un'entità esterna. Questo concetto risulta evidente osservando il diagramma UML dei casi d'uso in esame.



1.4 Modello dei casi d'uso

Tra i casi d'uso individuati, si è scelto di fornire una descrizione dettagliata dei primi due casi d'uso UC1 e UC2, i rimanenti verranno descritti nel formato informale.

1.4.1 UC1: Avviare una nuova partita

Nome del caso d'uso	**UC1: Avviare una nuova partita**
Portata	Applicazione DigiGoose!
Livello	Obiettivo utente

Attore primario	Giocatore
Parti interessate e interessi	Giocatore: vuole iniziare e gestire una partita del gioco dell'oca.
Precondizioni	L'applicazione è avviata.
Garanzia di successo	La partita viene avviata con successo e il sistema è pronto per ricevere le interazioni del giocatore.
Scenario principale di successo	1. Il giocatore seleziona l'opzione "Nuova Partita". 2. Il sistema visualizza le opzioni per impostare la partita (numero di giocatori, nome dei giocatori) 3. Il giocatore seleziona le opzioni desiderate. 4. Il giocatore conferma le impostazioni della partita. 5. Il sistema inizializza la partita con le impostazioni scelte. 6. Il sistema visualizza il tabellone di gioco, le pedine e i nomi dei giocatori. 7. Il sistema determina quale giocatore inizia la partita.
Estensioni	1. In qualsiasi momento, l'applicazione si blocca: 1. Il giocatore riavvia l'applicazione. 2. Il giocatore seleziona opzioni di gioco non valide: 1. Il sistema visualizza un messaggio di errore. 2. Il giocatore modifica le opzioni e riprova. 3. Il giocatore annulla la creazione della partita: 1. Il sistema torna alla schermata principale.
Requisiti speciali	Interfaccia utente intuitiva per la selezione delle opzioni di gioco.
Elenco delle varianti tecnologiche e dei dati	Dati: impostazioni della partita (numero di giocatori, nomi dei giocatori)
Frequenza delle ripetizioni	Ogni volta che un giocatore vuole giocare.
Varie	

1.4.2 UC2: Tirare i dadi

Nome del caso d'uso	**UC2: Tirare i dadi**
Portata	Applicazione DigiGoose!
Livello	Obiettivo utente
Attore primario	Giocatore
Parti interessate e interessi	Giocatore: Vuole lanciare i dadi per determinare il movimento della sua pedina. Il gioco deve essere equo e seguire le regole.
Precondizioni	Una partita è in corso ed è il turno del giocatore.

Garanzia di successo	Il sistema genera un risultato valido (2-12) del lancio dei dadi e lo visualizza al giocatore.
Scenario principale di successo	1. Il giocatore seleziona l'azione "Lancia i dadi". 2. Il sistema genera due numeri casuali tra 1 e 6. 3. Il sistema visualizza il risultato della somma dei due numeri. 4. Il sistema entra nel caso d'uso UC3 e viene mossa automaticamente la pedina del giocatore del numero di caselle corrispondente al risultato del lancio dei dadi. Il sistema controlla se la pedina del giocatore si trova nella casella 63, se sì, avvia UC6. 5. Il sistema entra nel caso d'uso UC4 e applica l'effetto della casella speciale corrispondente (se speciale). 6. Il sistema entra nel caso d'uso UC5 e passa il turno al prossimo giocatore.
Estensioni	a) L'applicazione non risponde al comando di tirare i dadi: 1. Il giocatore riavvia l'applicazione.
Requisiti speciali	Animazione per simulare il lancio dei dadi.
Elenco delle varianti tecnologiche e dei dati	Dati: Risultato del lancio dei dadi (due numeri interi tra 1 e 6).
Frequenza delle ripetizioni	Ad ogni turno di un giocatore.
Varie	

1.4.3 UC3: Muovere la pedina

1. Il sistema, dopo aver tirato i dadi, seleziona la pedina del giocatore in questione.
2. Il sistema calcola la nuova posizione della pedina in base al risultato del lancio dei dadi dalla posizione corrente.
3. Il sistema sposta la pedina sulla nuova casella.
4. Se la nuova casella è la casella 63, avvia UC6.
5. Se la nuova casella supera la casella 63, muovi la pedina all'indietro della differenza di caselle.
6. Se la casella è una casella speciale, avvia UC4, altrimenti avvia UC5.

1.4.4 UC4: Gestire le caselle speciali

1. Il sistema determina da UC3 se la pedina è arrivata su una casella speciale.
2. Se la pedina è su una casella speciale, il sistema identifica il tipo di casella.
3. Il sistema applica la regola specifica della casella.
4. Il sistema aggiorna la posizione della pedina e lo stato del gioco.
5. Il sistema avvia UC5.

1.4.5 Gestire i turni

1. All'avvio della partita, viene stabilito casualmente l'ordine di gioco dei giocatori.
2. Dopo che un giocatore ha completato la sua azione (tirare i dadi e muovere la pedina), il sistema passa il controllo al giocatore successivo in base all'ordine generato prima.
3. Durante la partita, il sistema si assicura che ogni turno venga rispettato, verificando il completamento delle azioni richieste e, se necessario, gestendo eventuali situazioni anomale (come turni saltati o penalità) in linea con le regole di gioco.

1.4.6 UC6: Determinare il vincitore

1. Quando un giocatore raggiunge esattamente la casella 63, il sistema termina la partita e dichiara il giocatore vincitore.
2. Il sistema riporta il giocatore al menu principale, ripulendo il salvataggio per un'eventuale nuova partita.

1.5 Documento di visione

Questa è una bozza iniziale del documento di visione. Durante lo sviluppo verrà esteso e, in caso, allegato in appendice, se la lunghezza del documento dovesse richiederlo.

1.5.1 Introduzione

1.5.1.1 Scopo

L'obiettivo di questo progetto è sviluppare un'applicazione digitale del gioco dell'oca per digitalizzare l'esperienza di gioco del classico gioco da tavolo. L'applicazione si propone di ricreare le dinamiche del gioco tradizionale in un ambiente digitale interattivo.

1.5.1.2 Portata

Questo documento di visione riguarda il progetto DigiGoose. Mi occuperò di sviluppare una soluzione software che permetta agli utenti di giocare al gioco dell'oca sul proprio computer. Il sistema sarà sviluppato in linguaggio Java.

1.5.1.3 Definizioni, acronimi e abbreviazioni

- DigiGoose! Nome dell'applicazione software per il gioco dell'oca digitale.
- UC: Caso d'uso

1.5.1.4 Riferimenti

- Regole ufficiali del gioco dell'oca Clementoni (versione moderna)

1.5.2 Posizionamento

1.5.2.1 Opportunità di business

L'applicazione DigiGoose offre l'opportunità di digitalizzare un gioco da tavolo classico, rendendolo accessibile a un pubblico più ampio e moderno. Il gioco digitale può offrire funzionalità aggiuntive rispetto al gioco da tavolo classico, ossia come la possibilità di giocare da soli contro il computer.

1.5.2.2 Formulazione del problema

Attualmente, il gioco dell'oca richiede la presenza fisica di un tabellone e di più giocatori. Questo può limitare la possibilità di giocare in qualsiasi momento e luogo.

1.5.2.3 Formulazione della posizione del prodotto

Destinatari	Il prodotto è rivolto a tutti, grandi e piccoli, sia occasionali che esperti.
Obiettivi	L'obiettivo è fornire un'esperienza di gioco digitale fedele, che rispetti le regole del gioco tradizionale.
Tipologia	Prodotto software.
Funzione	DigiGoose offre una soluzione digitale per giocare al gioco dell'oca.
Soluzioni alternative attuali	Esistono già alcune versioni digitali del gioco dell'oca, ma spesso con funzionalità limitate.
Caratteristiche prodotto	Possibilità di personalizzare le pedine, opzione di giocare da soli o con altri giocatori, salvataggio e caricamento della partita in corso.

1.5.3 Parti interessate e descrizioni utente

1.5.3.1 Riepilogo delle parti interessate

Nome	Descrizione	Responsabilità
Sviluppatore	Responsabile dello sviluppo del prodotto	Progettazione e implementazione software

1.5.3.2 Riepilogo dell'utente

Nome	Descrizione	Responsabilità	Parte interessata
Giocatore	Utilizzatore dell'applicazione	Utilizza l'applicazione per giocare al gioco.	Sé stesso

1.5.3.3 Ambiente dell'utente

L'applicazione sarà utilizzata su personal computer. L'utente tipo avrà familiarità con l'uso di videogiochi e applicazioni digitali.

1.5.3.4 Utente (Giocatore)

Rappresentante	Giocatore di DigiGoose!
Descrizione	Appassionato di giochi da tavolo, interessato a giocare al gioco dell'oca in formato digitale.
Competenze	Conoscenza di base dell'uso di computer e videogiochi.
Responsabilità	Fornire riscontri sull'applicazione.
Criteri di Successo	Facilità d'uso, esperienza di gioco semplice e divertente.
Coinvolgimento	L'utente sarà coinvolto nella fase di test del software.

Elaborati aggiuntivi	
Commenti / Problemi	

1.5.3.5 Sviluppatore

Rappresentante	**Salvatore Emmanuel La Porta**
Descrizione	Sviluppatore software responsabile della realizzazione del prodotto.
Competenze	Buone capacità di programmazione e conoscenza dello sviluppo software.
Responsabilità	Analisi dei requisiti, progettazione e implementazione del software, testing del prodotto finale.
Criteri di Successo	Stabilità dell'applicazione, numero ridotto di bug.
Coinvolgimento	Lo sviluppatore è coinvolto in tutte le fasi della realizzazione del software.
Elaborati aggiuntivi	
Commenti / Problemi	

1.5.3.6 Alternative e concorrenza

- Applicazioni del gioco dell'oca per dispositivi mobili.
- Siti web che offrono versioni online del gioco dell'oca.

1.5.4 Descrizione generale del prodotto

1.5.4.1 Punto di vista del prodotto

L'applicazione sarà eseguita su personal computer.

1.5.4.2 Riepilogo dei vantaggi

Vantaggi per le parti interessate	**Caratteristica di supporto**
Accessibilità e portabilità	L'applicazione Java può essere eseguita su qualsiasi computer con una Java Virtual Machine (JVM) installata.
Single player/Multiplayer	L'opzione di giocare da soli o con altri giocatori offre flessibilità.
Continuazione del gioco	La funzionalità di salvataggio e caricamento della partita corrente permette ai giocatori di riprendere il gioco da dove lo avevano interrotto.

1.6 Regole di business

Le seguenti regole di business definiscono il comportamento del gioco dell'oca digitale DigiGoose:

ID	Regola	Modificabilità	Sorgente
R1	Il numero di caselle di cui una pedina si sposta è determinato dal risultato del lancio dei dadi.	Bassa – Lo scopo è ricreare l'esperienza di gioco fornita da Clementoni per la versione moderna.	Regolamento Clementoni
R2	Quando un giocatore arriva su una casella speciale si applicano le seguenti regole: • **5:** Fai un balzo e rilancia i dadi. • **6:** Il ponte accelera la tua corsa: prosegui fino alla casella 12. • **9:** Ritorna alla casella 1. • **14:** L'oca spicca il volo: raddoppia il punteggio dei dadi. • **18:** È l'ora del pasto: fermati un turno. • **19:** La locanda è costosa: rimani fermo per un turno. • **23:** Fai un bel volo: vai alla casella 28. • **26:** Per proseguire devi ottenere 3 o 6 con almeno un dado. • **27:** Rilancia i dadi e torna indietro per il numero di caselle indicato. • **31:** L'oca è caduta nel pozzo: resti fermo 2 turni. • **32:** Raddoppia il punteggio indicato dai dadi. • **36:** L'oca depone le uova: fermati un turno. • **41:** Rilancia i dadi e vai avanti. • **42:** Ti sei perso nel labirinto: ritorna alla casella 35. • **45:** Fai cinque passi indietro: torna alla casella 40. • **50:** Rilancia i dadi e vai avanti. • **52:** L'oca è in prigione: resti fermo 2 turni. • **53:** Per proseguire devi ottenere 4 o 5 con almeno un dado. • **54:** Voli fino alla casella 57, ma ti fermi un turno. • **58:** Ritorna alla casella 1. • **59:** Rilancia i dadi e torna indietro per il numero di caselle indicato.	Bassa – Lo scopo è ricreare l'esperienza di gioco fornita da Clementoni per la versione moderna.	Regolamento Clementoni

R3	Il gioco supporta da 2 a 6 giocatori. I giocatori agiscono a turno e l'ordine di gioco viene determinato casualmente all'inizio della partita.	Bassa – Struttura base definita per garantire il corretto svolgimento del turno di gioco.	Documentazione di progetto DigiGoose
R4	Il primo giocatore che raggiunge esattamente la casella 63 vince la partita. Se un giocatore supera la casella 63, deve tornare indietro del numero di caselle in eccesso e riprovare al turno successivo.	Bassa – Regola fondamentale per la definizione della vittoria nel gioco.	Regolamento Clementoni
R5	I giocatori devono poter salvare e chiudere la partita in qualsiasi momento. Il sistema deve permette di caricare una partita salvata, consentendo di riprendere il gioco dallo stesso punto in cui era stato interrotto.	Alta – Funzionalità progettata per essere flessibile e aggiornabile in base alle esigenze degli utenti.	Documentazione di progetto DigiGoose
R6	L'interfaccia deve essere chiara, intuitiva e facile da usare. Il tabellone di gioco è visualizzato con le caselle numerate e le caselle speciali evidenziate; inoltre, i giocatori possono inserire i propri nomi e scegliere il colore della propria pedina.	Alta – Personalizzabile per migliorare l'esperienza utente e adattabile a possibili aggiornamenti grafici o funzionali.	Documentazione di progetto DigiGoose
R7	Il gioco deve includere la possibilità di giocare contro avversari controllati dal computer. La CPU è in grado di lanciare i dadi, muovere le pedine e seguire tutte le regole del gioco.	Bassa - La CPU deve comunque seguire le regole dei giocatori umani.	Documentazione di progetto DigiGoose

1.7 Specifiche supplementari

Aspetti legali

- DigiGoose! sarà rilasciato con licenza open source GPL v3.

Vincoli di sviluppo del software

- Tutto il software verrà scritto tramite l'utilizzo di Java.
- La persistenza dei dati sulla partita corrente sarà salvata in un file di tipo JSON.

Vincoli hardware e software

- Per eseguire il software non servono particolari requisiti per il sistema operativo, purché sia installata la Java Virtual Machine.

1.8 Glossario

1. **Attore**: Entità esterna che interagisce con il sistema. Nel contesto di DigiGoose, il giocatore umano è l'attore principale.
2. **UC**: Caso d'uso, descrizione di un insieme di sequenze di azioni che un sistema esegue per produrre un risultato osservabile di valore per un attore.
3. **CPU**: Nel contesto del gioco, si riferisce all'intelligenza artificiale che controlla i giocatori non umani.
4. **DigiGoose!**: Nome dell'applicazione software che implementa il gioco dell'oca in formato digitale.
5. **Gioco dell'oca**: Gioco da tavolo tradizionale in cui i giocatori lanciano i dadi per muovere le pedine su un percorso con caselle speciali che influenzano il movimento.
6. **GPL v3**: GNU General Public License versione 3, una licenza di software libero che garantisce agli utenti la libertà di utilizzare, studiare, condividere e modificare il software.
7. **Java**: Linguaggio di programmazione orientato agli oggetti utilizzato per sviluppare DigiGoose.
8. **JVM**: Java Virtual Machine, ambiente di esecuzione per programmi Java che permette di eseguire lo stesso codice su piattaforme diverse.
9. **Single player**: Modalità di gioco che consente a un giocatore umano di giocare contro avversari controllati dal computer.
10. **Multiplayer**: Modalità di gioco che consente a più giocatori umani di partecipare alla stessa partita.
11. **Partita**: Sessione di gioco completa dall'inizio (posizionamento delle pedine sulla casella di partenza) fino alla determinazione del vincitore.
12. **Pedina**: Rappresentazione virtuale del giocatore sulla tavola di gioco, identificata da un colore scelto.
13. **Tabellone**: Rappresentazione grafica del percorso di gioco composto da 63 caselle numerate, alcune delle quali sono caselle speciali con effetti particolari.
14. **Turno**: Periodo di gioco durante il quale un giocatore può compiere azioni (lanciare i dadi, muovere la pedina).

2 – Elaborazione – Iterazione 1

2.1 Introduzione

Conclusa la fase di ideazione, si passa alla fase di elaborazione. Lo scopo di questa fase è di implementare il nucleo dell'architettura software di DigiGoose. In questa prima iterazione ci concentreremo sull'implementare lo scenario principale di successo del caso d'uso UC1: Avviare una nuova partita; Infine, implementeremo un caso d'uso di start up necessario per gestire l'inizializzazione per questa iterazione.

2.2 Aggiornamento caso d'uso UC1

Durante la lettura della parte sulla fase di Ideazione, sono emerse delle parti mancanti al caso d'uso UC1, al fine di poter rispettare i requisiti dell'applicazione. Verrà quindi riproposto il caso d'uso a seguire, con le modifiche aggiunte.

Nome del caso d'uso	**UC1: Avviare una nuova partita**
Portata	Applicazione DigiGoose!
Livello	Obiettivo utente
Attore primario	Giocatore
Parti interessate e interessi	Giocatore: vuole iniziare e gestire una partita del gioco dell'oca con un numero variabile di partecipanti, scegliendo tra giocatori umani e controllati dal computer, colori delle pedine e i loro nomi.
Precondizioni	L'applicazione è avviata.
Garanzia di successo	La partita viene avviata con successo, con il numero di giocatori desiderato (umani e computer), l'ordine del gioco determinato, le pedine col colore scelto per giocatore sulla casella iniziale e il sistema pronto a ricevere l'interazione del primo giocatore.
Scenario principale di successo	8. Il giocatore seleziona l'opzione "Nuova Partita". 9. Il sistema visualizza le opzioni per impostare la partita, ossia: a. Numero di giocatori (2 – 6); b. Scelta tra umano e computer per ogni giocatore c. Inserimento del nome di ciascun giocatore umano d. Scelta del colore della pedina; 10. Il giocatore seleziona il numero di giocatori desiderato e configura se è un umano o computer per ciascuno. 11. Per i giocatori umani, il giocatore inserisce i nomi nei campi appositi. Il sistema valida che il nome non sia vuoto o contiene caratteri non validi. 12. Il giocatore seleziona un colore univoco per la propria pedina. Il sistema assicura che ogni giocatore abbia un colore diverso. 13. Il giocatore conferma le impostazioni della partita. 14. Il sistema inizializza la partita con le impostazioni scelte. 15. Il sistema determina casualmente quale giocatore inizia la partita.

	16. Il sistema visualizza il tabellone di gioco, le pedine posizionate sulla freccia che indica la casella numero 1, e i nomi dei giocatori, indicando il giocatore a cui tocca il suo turno. Il sistema aggiorna sessione corrente (non verrà applicata in questa elaborazione, ma la creazione di una nuova partita deve sovrascrivere la sessione salvata per "riprendi partita")
Estensioni	1) In qualsiasi momento, l'applicazione si blocca: a) Il giocatore riavvia l'applicazione. 2) Il giocatore seleziona opzioni di gioco non valide: a) Il sistema visualizza un messaggio di errore. b) Il giocatore modifica le opzioni e riprova. 3) Il giocatore annulla la creazione della partita: a) Il sistema torna alla schermata principale.
Requisiti speciali	1) Interfaccia utente intuitiva per la selezione delle opzioni di gioco. 2) Il sistema deve assicurare l'univocità del colore delle pedine. 3) Il sistema deve fornire un feedback chiaro in caso di input non validi.
Elenco delle varianti tecnologiche e dei dati	Dati: impostazioni della partita (numero di giocatori, nomi dei giocatori, tipo di giocatori, colori delle pedine) Metodi: selezione del primo giocatore casuale, la posizione delle pedine deve essere alla freccia prima della casella 1.
Frequenza delle ripetizioni	Ogni volta che un giocatore vuole iniziare una nuova partita.
Varie	

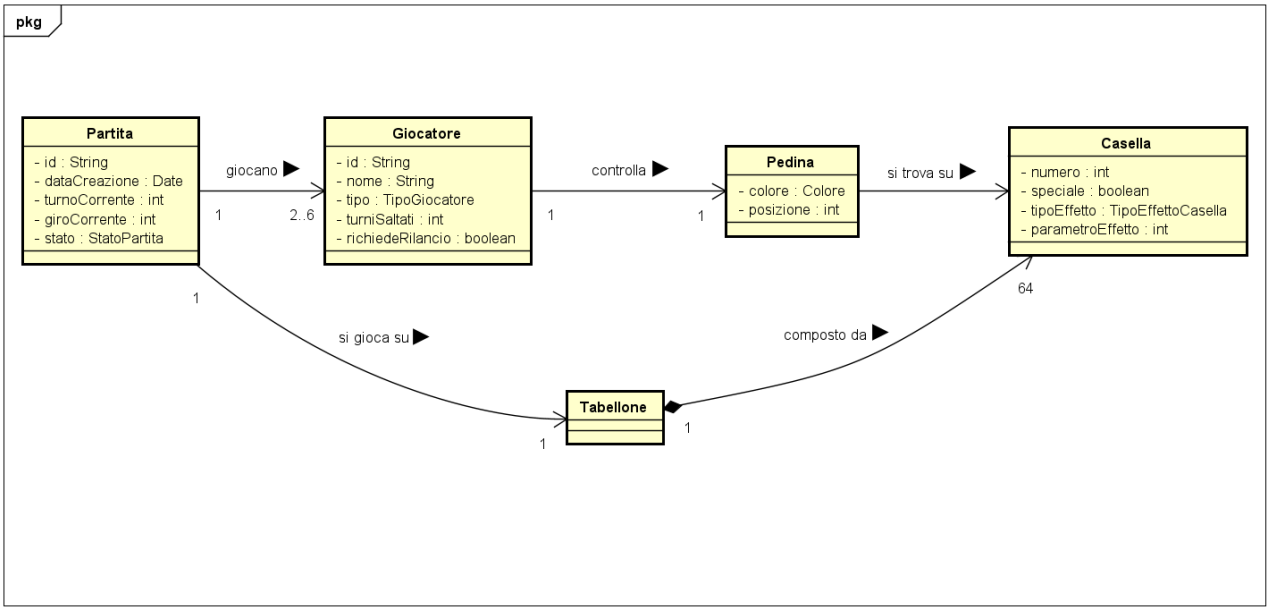
2.3 Analisi orientata agli oggetti

Verrà adesso eseguita l'analisi OO tramite i seguenti strumenti: modello di dominio, diagramma di sequenza SSD del sistema e contratti delle operazioni.

2.2.1 Modello di dominio

Dopo l'analisi del caso d'uso, sono stati scelti le seguenti classi concettuali per ottenere lo scenario principale di successo:

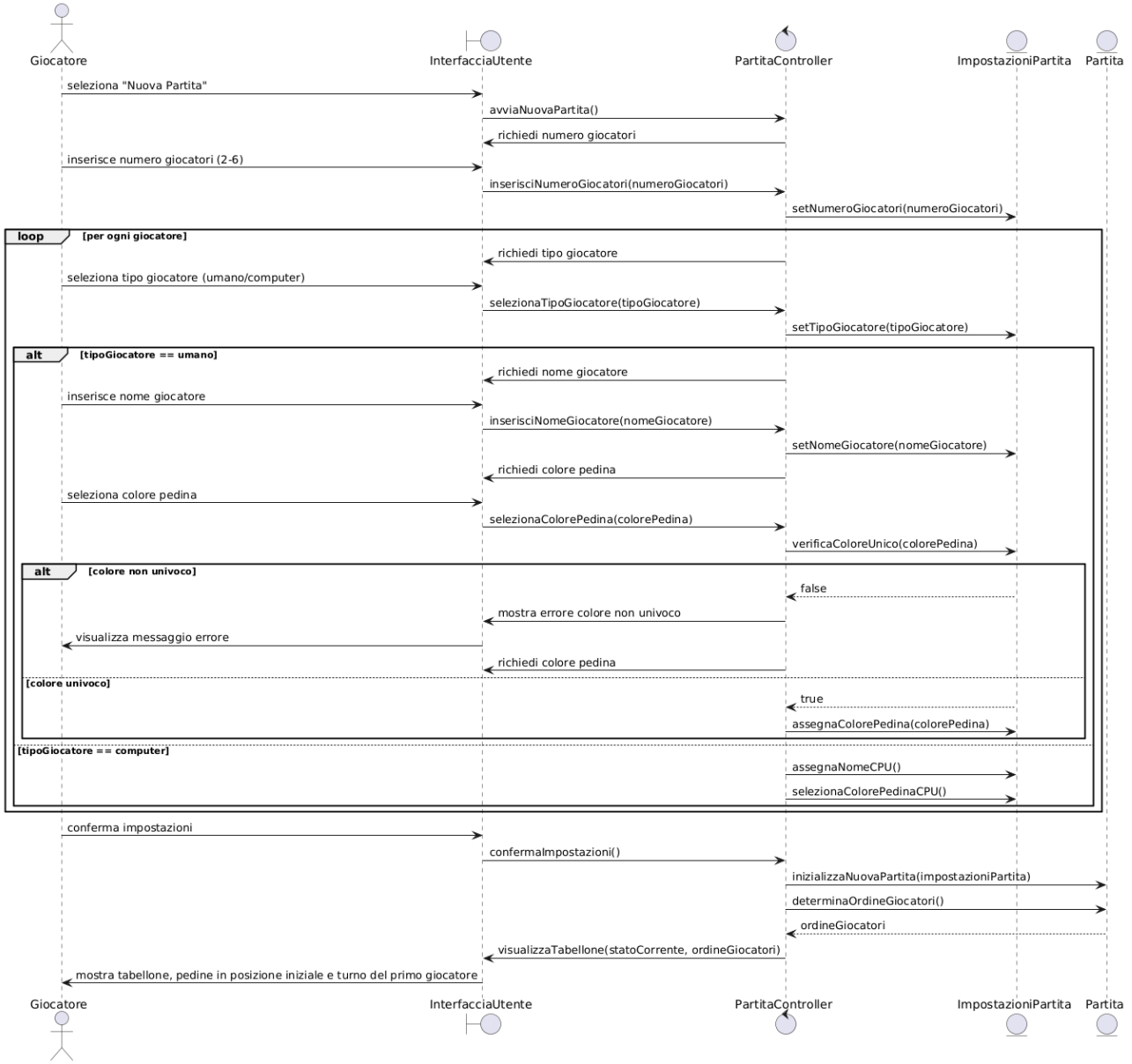
- **Giocatore**: rappresenta un partecipante al gioco, rappresentato da una pedina nel tabellone;
- **Partita**: Rappresenta la partita in corso, dove sono presenti tutti gli elementi del gioco;
- **Pedina**: Rappresenta un giocatore sul tabellone.
- **Tabellone**: Rappresenta il tabellone di gioco, composto da caselle;
- **Casella**: Rappresenta una singola casella sul tabellone;



2.2.2 Diagramma di sequenza di sistema

Il passo successivo, adesso, è la creazione del diagramma di sequenza SSD del sistema, con lo scopo di illustrare il corso degli eventi di input e output per lo scenario principale di successo. Il diagramma conterrà le seguenti interazioni, spiegate in breve:

1. L'utente inizia il processo di avviare una nuova partita all'interno dell'applicazione.
2. Il sistema risponde, chiedendo il numero di giocatori che parteciperanno alla partita.
3. L'utente inserisce il numero di giocatori desiderato e lo comunica al sistema.
4. Per ogni giocatore:
    a. Il sistema chiede il tipo di giocatore (umano o computer).
    b. L'utente seleziona il tipo desiderato per quel giocatore.
    c. Se il giocatore è un umano:
        i. Il sistema richiede il nome del giocatore.
        ii. Il giocatore inserisce il proprio nome.
        iii. Il sistema richiede il colore della pedina desiderato dal giocatore.
        iv. Il giocatore seleziona il colore della pedina.
    d. Se il giocatore è controllato dal computer:
        i. Il sistema assegna automaticamente un nome predefinito (CPU 1, CPU 2,...) assicurandosi sia univoco.
        ii. Il sistema seleziona automaticamente un colore per la pedina casuale tra quelli ancora disponibili.
5. Il sistema esegue internamente la funzione di inizializzazione con le impostazioni scelte.
6. Il sistema determina internamente l'ordine dei giocatori in maniera casuale.
7. Il sistema comunica l'ordine di gioco, visualizza il tabellone di gioco con le pedine nella posizione iniziale e indica il primo giocatore.



2.2.3 Contratti delle operazioni

Adesso vengono definiti i contratti, per definire le principali operazioni di sistema al fine di gestire quanto visto nell'SSD:

Contratto CO1: avviaNuovaPartita

Operazione:	avviaNuovaPartita()
Riferimenti:	caso d'uso: Avviare una nuova partita

Pre-condizioni:	L'applicazione DigiGoose! è in esecuzione e l'Attore Primario (Giocatore) ha selezionato l'opzione "Avvia nuova partita".
Post-condizioni:	Il sistema è nello stato di attesa dell'inserimento del numero di giocatori.

Contratto CO2: inserisciNumeroGiocatori

Operazione:	inserisciNumeroGiocatori(numeroGiocatori: integer)
Riferimenti:	caso d'uso: Avviare una nuova partita
Pre-condizioni:	Il sistema ha richiesto il numero di giocatori.
Post-condizioni:	• Il sistema è nello stato di attesa della selezione del tipo di giocatore per il primo giocatore (umano o computer). • L'attributo 'numeroGiocatori' della sessione di nuova partita è impostato al valore fornito. • Si avvia un ciclo per raccogliere le informazioni di ogni giocatore.

Contratto CO3: selezionaTipoGiocatore

Operazione:	selezionaTipoGiocatore(tipoGiocatore: enum {umano, computer})
Riferimenti:	caso d'uso: Avviare una nuova partita
Pre-condizioni:	Il sistema ha richiesto il tipo di giocatore per il giocatore corrente
Post-condizioni:	• Se tipoGiocatore è 'umano', il sistema è nello stato di attesa dell'inserimento del nome del giocatore. • Se tipoGiocatore è 'computer', il sistema procede automaticamente all'assegnazione del nome CPU e alla selezione del colore della pedina. • L'attributo 'tipo' del giocatore corrente nella sessione di nuova partita è impostato al valore fornito.

Contratto CO4: inserisciNomeGiocatore

Operazione:	inserisciNomeGiocatore(nomeGiocatore: string)
Riferimenti:	caso d'uso: Avviare una nuova partita
Pre-condizioni:	Il sistema ha richiesto il nome del giocatore (giocatore umano)
Post-condizioni:	• Il sistema è nello stato di attesa della selezione del colore della pedina per il giocatore corrente. • L'attributo 'nome' del giocatore umano corrente nella sessione di nuova partita è impostato al valore fornito.

Contratto CO5: selezionaColorePedina

Operazione:	selezionaColorePedina(colorePedina: enum {elenco dei colori disponibili})
Riferimenti:	caso d'uso: Avviare una nuova partita
Pre-condizioni:	l sistema ha richiesto il colore della pedina per il giocatore corrente (giocatore umano).
Post-condizioni:	• L'attributo 'colorePedina' del giocatore umano corrente viene verificato per unicità tramite verificaColoreUnico. • Se il colore è univoco, l'attributo 'colorePedina' del giocatore viene impostato al valore fornito. • Se il colore non è univoco, il sistema mostra un messaggio di errore e richiede nuovamente la selezione del colore. • Se ci sono altri giocatori da configurare, il sistema richiede il tipo di giocatore successivo, altrimenti procede con la conferma delle impostazioni.

Contratto CO6: verificaColoreUnico

Operazione:	verificaColoreUnico(colorePedina: enum {elenco dei colori disponibili})
Riferimenti:	caso d'uso: Avviare una nuova partita
Pre-condizioni:	È stato selezionato un colore per il giocatore corrente.
Post-condizioni:	• Il sistema ha verificato se il colore selezionato è già stato assegnato ad un altro giocatore. • Ritorna true se il colore è disponibile (univoco), false altrimenti.

Contratto CO7: assegnaNomeCPU

Operazione:	assegnaNomeCPU()
Riferimenti:	caso d'uso: Avviare una nuova partita
Pre-condizioni:	Il tipo di giocatore corrente è 'computer'.
Post-condizioni:	• Il sistema ha generato un nome univoco per il giocatore computer (ad esempio, "CPU 1", "CPU 2", ...). • L'attributo 'nome' del giocatore computer corrente nella sessione di nuova partita è impostato sul nome generato. • Il sistema procede alla selezione automatica del colore della pedina per il giocatore computer.

Contratto CO8: selezionaColorePedinaCPU

Operazione:	selezionaColorePedinaCPU()
Riferimenti:	caso d'uso: Avviare una nuova partita
Pre-condizioni:	Il tipo di giocatore corrente è 'computer' e il nome è stato assegnato.
Post-condizioni:	• Il sistema ha selezionato un colore della pedina casuale tra quelli disponibili (non scelti dai giocatori umani e da altri giocatori computer). • L'attributo 'colorePedina' del giocatore computer corrente nella sessione di nuova partita è impostato sul colore selezionato. • Se ci sono altri giocatori da configurare, il sistema richiede il tipo di giocatore successivo. Altrimenti, procede all'inizializzazione della partita.

Contratto CO9: confermaImpostazioni

Operazione:	confermaImpostazioni()
Riferimenti:	caso d'uso: Avviare una nuova partita
Pre-condizioni:	Sono state raccolte tutte le informazioni per tutti i giocatori (tipo, nome, colore pedina).
Post-condizioni:	Il sistema procede all'inizializzazione della nuova partita con le impostazioni specificate.

Contratto CO10: inizializzaNuovaPartita

Operazione:	inizializzaNuovaPartita(impostazioniPartita: collezione di informazioni sui giocatori)
Riferimenti:	caso d'uso: Avviare una nuova partita
Pre-condizioni:	Le impostazioni della partita sono state confermate.
Post-condizioni:	• È stata creata una nuova istanza di 'Partita'. • Per ogni giocatore specificato, è stata creata una nuova istanza di 'Giocatore' (umano o computer) con i rispettivi attributi (nome, colorePedina) e associata alla 'Partita'. • È stata gestita l'associazione univoca dei colori delle pedine ai giocatori. • La 'Partita' è nello stato iniziale.

Contratto CO11: determinaOrdineGiocatori

Operazione:	determinaOrdineGiocatori()
Riferimenti:	caso d'uso: Avviare una nuova partita
Pre-condizioni:	La nuova partita è stata inizializzata con i giocatori.
Post-condizioni:	• È stato selezionato casualmente l'ordine dei giocatori della partita. • L'attributo 'giocatoreCorrente' della 'Partita' è impostato sul primo giocatore seguendo l'ordine. • L'ordine dei giocatori viene restituito al controller.

Contratto CO12: visuaizzaTabellone

Operazione:	visualizzaTabellone(statoCorrente: stato del gioco, ordineGiocatori: [Giocatori])
Riferimenti:	caso d'uso: Avviare una nuova partita
Pre-condizioni:	È stato determinato l'ordine dei giocatori.
Post-condizioni:	• L'interfaccia utente visualizza il tabellone di gioco con le pedine nella posizione iniziale (sulla freccia che indica la casella numero 1). • L'interfaccia utente visualizza i nomi di tutti i giocatori partecipanti. • L'interfaccia utente indica chiaramente il nome del giocatore a cui tocca il turno, seguendo l'ordine determinato.

	• Il sistema aggiorna la sessione corrente, sovrascrivendo eventuali partite salvate precedentemente.

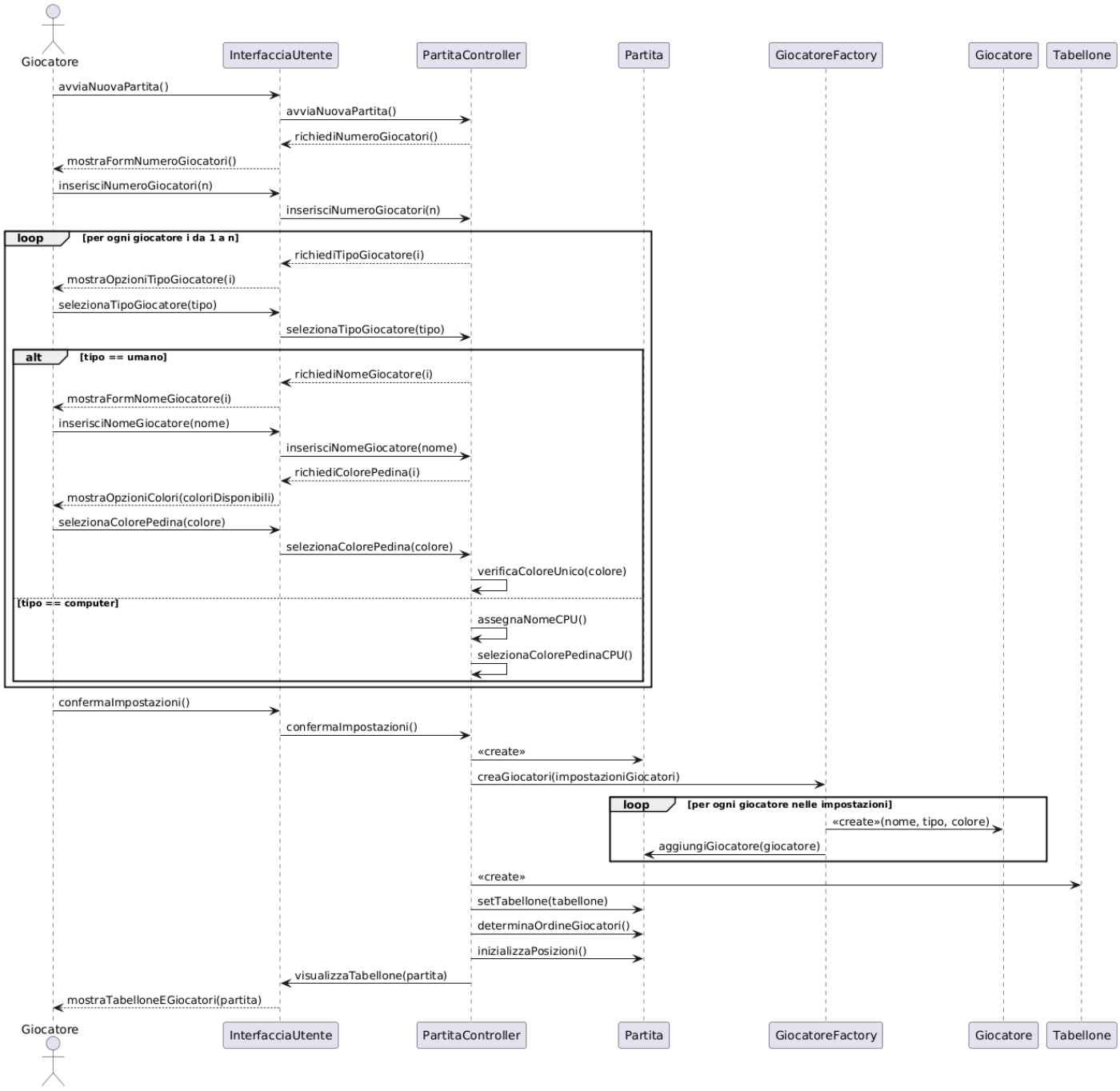
2.4 Progettazione

Adesso concentriamoci su gli oggetti software necessari per la progettazione. L'obiettivo di questa fase è il modello di progetto, che illustreremo attraverso i diagrammi di sequenza e delle classi.

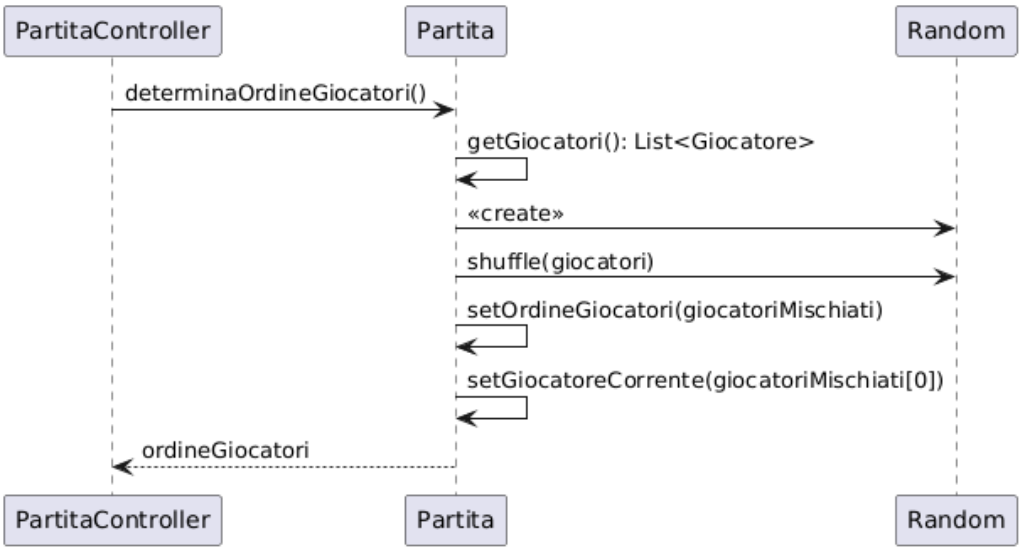
2.4.1 Diagrammi di sequenza

Il diagramma di sequenza ha lo scopo di illustrare nel dettaglio il flusso degli eventi che si verificano tra il giocatore e il sistema.

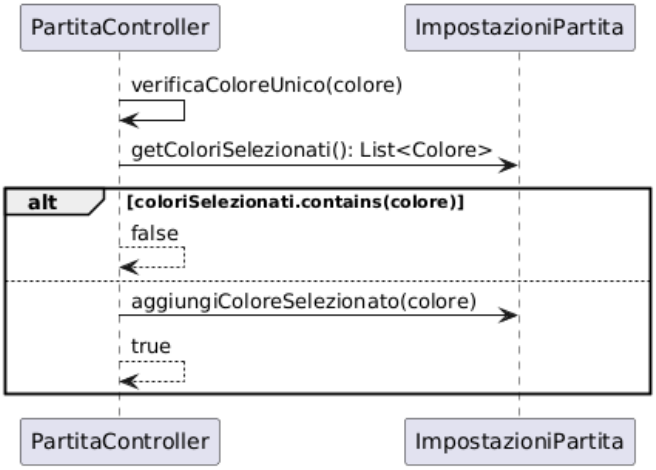
- Avviare una nuova partita



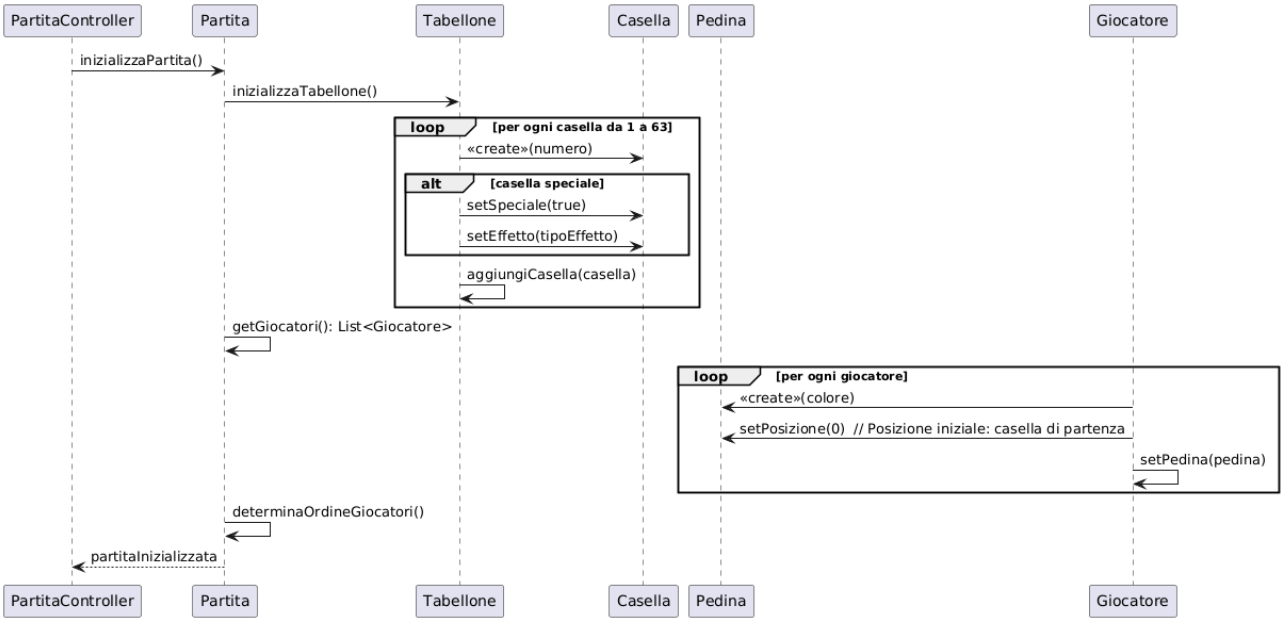
- Determinare l’ordine dei giocatori



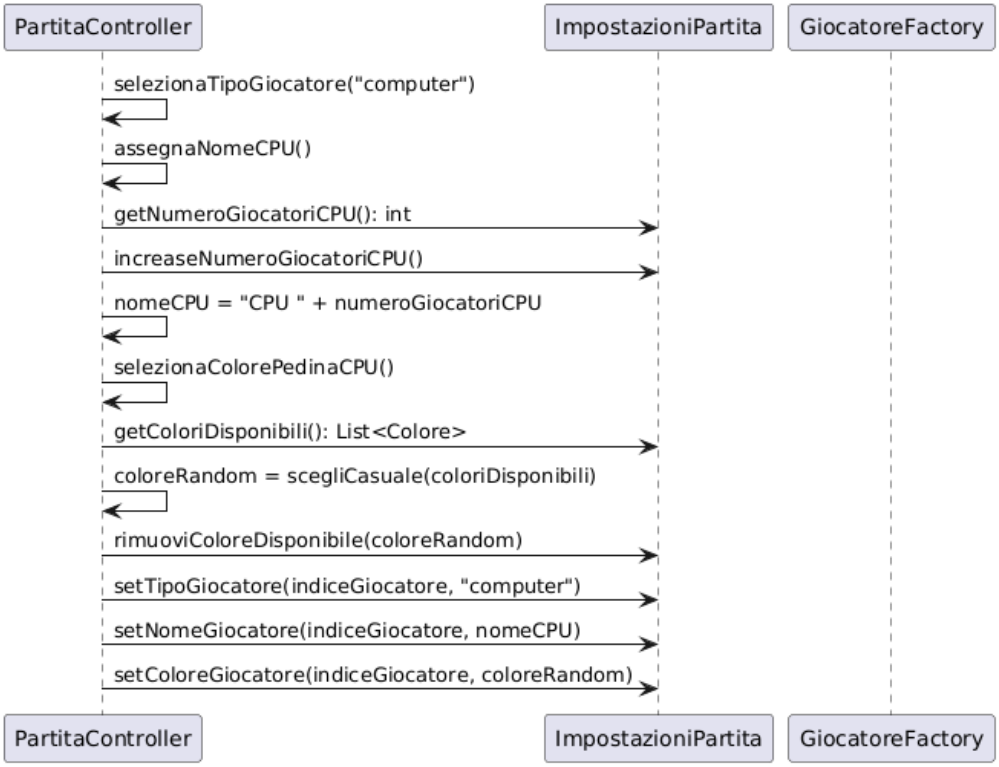
- Verifica dell'unicità del colore della pedina



- Inizializzazione della partita



- Gestione dei giocatori controllati dal computer



2.4.2 Diagramma delle classi

Infine, facciamo il diagramma delle classi, strutturandolo secondo l'architettura MVC. Per una maggior fruibilità nella documentazione viene fornito l'SVG del diagramma delle classi.

3 – Elaborazione – Iterazione 2

3.1 Introduzione

In questa iterazione della fase di elaborazione sono stati implementati tutti gli scenari di successo di tutti i casi d'uso rimanenti, da UC2 a UC6 e le estensioni di tutti i casi d'uso.

3.2 Casi d'uso in dettaglio

Di seguito verranno presentati nuovamente tutti i casi d'uso, in dettaglio.

3.2.1 UC1: Avviare una nuova partita

Nome del caso d'uso	**UC1: Avviare una nuova partita**
Portata	Applicazione DigiGoose!
Livello	Obiettivo utente
Attore primario	Giocatore
Parti interessate e interessi	Giocatore: vuole iniziare e gestire una partita del gioco dell'oca con un numero variabile di partecipanti, scegliendo tra giocatori umani e controllati dal computer, colori delle pedine e i loro nomi.
Precondizioni	L'applicazione è avviata.
Garanzia di successo	La partita viene avviata con successo, con il numero di giocatori desiderato (umani e computer), l'ordine del gioco determinato, le pedine col colore scelto per giocatore alla posizione iniziale (0) e il sistema pronto a ricevere l'interazione del primo giocatore.
Scenario principale di successo	17. Il giocatore seleziona l'opzione "Nuova Partita". 18. Il sistema visualizza le opzioni per impostare la partita, ossia: a. Numero di giocatori (2 – 6); b. Scelta tra umano e computer per ogni giocatore c. Inserimento del nome di ciascun giocatore umano d. Scelta del colore della pedina; 19. Il giocatore seleziona il numero di giocatori desiderato e configura se è un umano o computer per ciascuno. 20. Per i giocatori umani, il giocatore inserisce i nomi nei campi appositi. Il sistema valida che il nome non sia vuoto o contiene caratteri non validi. 21. Il giocatore seleziona un colore univoco per la propria pedina. Il sistema assicura che ogni giocatore abbia un colore diverso. 22. Il giocatore conferma le impostazioni della partita. 23. Il sistema inizializza la partita con le impostazioni scelte. 24. Il sistema determina casualmente quale giocatore inizia la partita. 25. Il sistema visualizza il tabellone di gioco, le pedine posizionate alla casella 0, e i nomi dei giocatori, indicando il giocatore a cui tocca il suo turno. Il sistema aggiorna sessione corrente (non verrà applicata in questa elaborazione, ma la creazione di una nuova partita deve sovrascrivere la sessione salvata per "riprendi partita")

Estensioni	4) In qualsiasi momento, l'applicazione si blocca: a) Il giocatore riavvia l'applicazione. 5) Il giocatore seleziona opzioni di gioco non valide: a) Il sistema visualizza un messaggio di errore. b) Il giocatore modifica le opzioni e riprova. 6) Il giocatore annulla la creazione della partita: a) Il sistema torna alla schermata principale.
Requisiti speciali	4) Interfaccia utente intuitiva per la selezione delle opzioni di gioco. 5) Il sistema deve assicurare l'univocità del colore delle pedine. 6) Il sistema deve fornire un feedback chiaro in caso di input non validi.
Elenco delle varianti tecnologiche e dei dati	Dati: impostazioni della partita (numero di giocatori, nomi dei giocatori, tipo di giocatori, colori delle pedine) Metodi: selezione del primo giocatore casuale, la posizione delle pedine deve essere sulla casella 0.
Frequenza delle ripetizioni	Ogni volta che un giocatore vuole iniziare una nuova partita.
Varie	L'implementazione corrente include codice per il salvataggio/caricamento (commentato in GestoreSalvataggio), ma questa funzionalità non è esposta nell'interfaccia utente e una nuova partita sovrascrive effettivamente qualsiasi stato precedente nella logica attuale della UI.

3.2.2 UC2: Tirare i dadi

Nome del caso d'uso	**UC2: Tirare i dadi**
Portata	Partita in corso
Livello	Sotto-funzione (parte del turno di un giocatore)
Attore primario	Giocatore corrente
Parti interessate e interessi	Giocatore corrente: vuole ottenere il risultato del lancio per determinare il movimento. Altri giocatori: il risultato del lancio potrebbe sbloccare condizioni che li tengono fermi. Sistema: ha bisogno del risultato per calcolare il movimento e applicare effetti.
Precondizioni	La partita è in stato IN_CORSO. È il turno del giocatore corrente. Il giocatore corrente non è bloccato per un numero fisso di turni (getTurniSaltati() > 0). Può essere bloccato da una condizione (getTurniSaltati() < 0).
Garanzia di successo	I dadi vengono simulati e lanciati. Il risultato è determinato e visualizzato. Eventuali condizioni di blocco di giocatori (incluso il

	giocatore corrente se bloccato condizionalmente) che dipendono dal risultato del lancio vengono verificate e rimosse se soddisfatte.
Scenario principale di successo	1. È il turno del giocatore corrente. 2. Il giocatore: a. Se è umano, il sistema gli chiede di lanciare i dadi. b. Se è CPU, il sistema procede autonomamente con il lancio, chiedendo conferma all'utente per il passaggio al turno successivo. 3. Il sistema simula il lancio di 2 dadi a 6 facce. 4. Il sistema comunica il risultato di ciascun dado e la loro somma al giocatore/utente. 5. Il sistema verifica se il risultato del lancio soddisfa le condizioni di sblocco per qualsiasi giocatore bloccato (getTurniSaltati() < 0, es. attesa di un 6, attesa di 3/6, attesa di 4/5, prigione). 6. Per ogni giocatore le cui condizioni di blocco sono soddisfatte dal lancio, il sistema rimuove lo stato di blocco (setTurniSaltati(0)) e informa l'utente (se umano). 7. Il giocatore corrente: a. Se era bloccato condizionalmente e *non* è stato sbloccato da questo lancio, il suo turno finisce qui senza muoversi. La gestione del passaggio al giocatore successivo avverrà in UC5. b. Se era libero o è stato sbloccato da questo lancio, il sistema procederà al movimento - UC3).
Estensioni	1. Il giocatore umano sceglie un'opzione diversa dal lanciare i dadi (es. Esci): a. L'UC termina e il flusso passa alla gestione dell'opzione scelta (es. ritorno al menu principale). 2. Si verifica un errore tecnico durante la simulazione del lancio dei dadi: a. Il sistema si blocca o gestisce l'errore (rif. UC1 Estensione 1).
Requisiti speciali	Il risultato dei dadi deve essere generato in modo casuale. Il sistema deve visualizzare chiaramente il risultato del lancio. Il sistema deve verificare e gestire le condizioni di sblocco per i giocatori bloccati (turniSaltati < 0) basate sul risultato del lancio.
Elenco delle varianti tecnologiche e dei dati	Dati: valore dei due dadi. Metodi: generazione casuale dei dadi (Dadi.lancia(), Random), verifica condizioni di sblocco (PartitaController.verificaELiberaGiocatoriBloccati()).
Frequenza delle ripetizioni	Una volta per ogni turno di un giocatore, a meno che non debba saltare il turno per un numero fisso (turniSaltati > 0).
Varie	

3.2.3 UC3: Muovere la pedina

Nome del caso d'uso	UC3: Muovere la pedina
Portata	Partita in corso
Livello	Sotto-funzione (parte del turno di un giocatore)
Attore primario	Sistema (ma agisce per conto di Giocatore)
Parti interessate e interessi	Giocatore corrente: vede la sua pedina avanzare sul tabellone. Tabellone: ospita la pedina nella sua nuova posizione.
Precondizioni	UC2 (Tirare i dadi) è stato completato con successo. Il giocatore corrente non è bloccato da una condizione (getTurniSaltati() >= 0 dopo la verifica in UC2). Il risultato totale del lancio dei dadi è disponibile.
Garanzia di successo	La pedina del giocatore corrente viene spostata sulla casella calcolata in base al risultato dei dadi. La nuova posizione rispetta le regole di gioco, inclusa la gestione del superamento della casella finale (63) tramite "rimbalzo". Il sistema identifica la casella di destinazione.
Scenario principale di successo	1. 1. Il sistema ottiene la somma dei valori dei dadi lanciati in UC2. 2. Il sistema ottiene la posizione corrente della pedina del giocatore corrente. 3. Il sistema calcola la potenziale nuova posizione sommando la posizione corrente al totale dei dadi. 4. Il sistema verifica se la potenziale nuova posizione supera la casella finale (63): a. Se la potenziale nuova posizione *non* supera 63, la nuova posizione è quella calcolata al passo 3. b. Se la potenziale nuova posizione *supera* 63, il sistema calcola la nuova posizione "rimbalzando" indietro da 63 del numero di caselle in eccesso rispetto a 63. 5. Il sistema aggiorna la posizione della pedina del giocatore alla nuova posizione finale calcolata. 6. Il sistema identifica la casella corrispondente alla nuova posizione sul tabellone. 7. Il sistema comunica la nuova posizione del giocatore all'utente. 8. La gestione degli effetti speciali di questa casella avviene in UC4.
Estensioni	1. La somma dei dadi non è disponibile o valida. a. UC3 non può essere eseguito. Si verifica un errore di sistema ancora da gestire.
Requisiti speciali	La logica di movimento deve gestire correttamente il "rimbalzo" oltre la casella 63.

Elenco delle varianti tecnologiche e dei dati	Dati: posizione corrente della pedina, somma dei dadi. Metodi: calcolo nuova posizione, aggiornamento posizione pedina.
Frequenza delle ripetizioni	Una volta per ogni turno giocato (non saltato per turniSaltati > 0 e sbloccato se turniSaltati < 0).
Varie	

3.2.4 UC4: Gestire le caselle speciali

Nome del caso d'uso	**UC4: Gestire le caselle speciali**
Portata	Partita in corso
Livello	Sotto-funzione (parte del turno di un giocatore)
Attore primario	Sistema (ma agisce per conto di Giocatore)
Parti interessate e interessi	Giocatore corrente: subisce o beneficia dell'effetto della casella. Tabellone: contiene le definizioni delle caselle speciali. Altri giocatori: potrebbero essere influenzati indirettamente (es. effetto che fa saltare un turno).
Precondizioni	UC3 (Muovere la pedina) è stato completato con successo. La pedina del giocatore corrente è arrivata su una casella. Il risultato del lancio dei dadi del turno corrente è disponibile.
Garanzia di successo	Se la casella su cui è arrivato il giocatore è una casella speciale (diversa dalla casella 63), l'effetto associato viene applicato al giocatore (e potenzialmente ad altri) secondo le regole del gioco. Lo stato o la posizione del giocatore possono essere modificati dall'effetto. Il sistema comunica l'effetto all'utente.
Scenario principale di successo	1. Il sistema identifica la casella di destinazione sulla quale è arrivata la pedina del giocatore corrente. 2. Il sistema verifica se la casella è la casella finale (63): a. Se sì, viene avviato UC6. b. Se la casella *non* è la casella 63, il sistema verifica se la casella è speciale. 3. Se la casella: a. *non* è speciale, UC4 termina. b. Se è speciale, il sistema determina il tipo di effetto. 4. Il sistema applica l'effetto base associato al tipo di casella speciale. Questo potrebbe modificare la posizione o lo stato del giocatore. 5. Il sistema verifica se l'effetto richiede un'ulteriore gestione basata sul risultato dei dadi (es. Raddoppia Movimento). 6. Se l'effetto richiede l'interazione con i dadi, il sistema applica la logica specifica utilizzando il risultato dei dadi del turno

	corrente. Questo potrebbe comportare un ulteriore spostamento della pedina. 7. Il sistema comunica il tipo di effetto applicato e la potenziale nuova posizione del giocatore all'utente.
Estensioni	1. La casella di destinazione non è valida. a. UC4 non può essere eseguito correttamente. Si verifica un errore di sistema da gestire. 2. Si verifica un errore durante l'applicazione della logica dell'effetto speciale. a. Il sistema gestisce l'errore (rif. UC1 Estensione 1).
Requisiti speciali	Deve essere implementata correttamente la logica per ogni tipo di casella speciale. La posizione o lo stato del giocatore devono essere aggiornati secondo l'effetto. Gli effetti che dipendono dal lancio dei dadi devono utilizzare il risultato corretto.
Elenco delle varianti tecnologiche e dei dati	Dati: tipo di effetto casella, posizione giocatore, stato giocatore, risultato dadi. Metodi: applicazione effetti, accesso casella.
Frequenza delle ripetizioni	Una volta per ogni turno giocato (non saltato per turniSaltati > 0 e sbloccato se turniSaltati < 0), se la casella di destinazione è speciale (diversa dalla 63).
Varie	

3.2.5 UC5: Gestire i turni

Nome del caso d'uso	**UC5: Gestire i turni**
Portata	Partita in corso
Livello	Sistema (coordinamento del flusso di gioco)
Attore primario	Sistema
Parti interessate e interessi	Tutti i giocatori: attendono il proprio turno e osservano lo stato del gioco.
Precondizioni	La partita è in stato IN_CORSO. Il turno del giocatore precedente è terminato (ha completato il movimento e gli effetti, o ha saltato il turno per turniSaltati > 0).
Garanzia di successo	Il sistema identifica correttamente il giocatore successivo nell'ordine stabilito. Se il giocatore successivo deve saltare il turno per un numero fisso (turniSaltati > 0), questo turno viene gestito come "saltato" e il sistema passa immediatamente al giocatore ancora successivo, aggiornando il conteggio dei turni da saltare per quel giocatore.

	Il giro e il turno vengono incrementati correttamente all'inizio di ogni nuovo giro e per ogni giocatore effettivo. Il sistema presenta l'interfaccia/logica per il giocatore corrente che può agire.
Scenario principale di successo	1. Il turno del giocatore corrente (precedente) si conclude. 2. Il sistema verifica se esso è rimasto bloccato condizionalmente in UC2 o UC4. 3. Se *non* è bloccato condizionalmente, il sistema identifica il giocatore successivo nell'ordine di gioco predefinito. 4. Il sistema verifica se questo prossimo giocatore deve saltare il turno per un numero fisso. 5. Se il prossimo giocatore deve saltare il turno: a. Il sistema informa l'utente che il giocatore salta il turno. b. Il sistema decrementa il contatore dei turni da saltare per quel giocatore. c. l sistema ripete i passi 3 e 4 per trovare il giocatore successivo nell'ordine (questo ciclo continua finché non si trova un giocatore che *non* deve saltare il turno). 6. Se il prossimo giocatore *non* deve saltare il turno per un numero fisso, questo giocatore diventa il giocatore corrente per il nuovo turno. 7. Il sistema incrementa il contatore del turno. Se questo giocatore è il primo nell'ordine del giro, il sistema incrementa il contatore del giro. 8. Il sistema presenta l'interfaccia utente/logica per consentire al giocatore corrente di eseguire il suo turno (avvia UC2). 9. Se il giocatore precedente *era* bloccato condizionalmente, rimane il giocatore corrente per la prossima iterazione del ciclo di gioco principale, e il flusso riprende da UC2 per lui, senza passare al giocatore successivo nell'ordine finché non viene sbloccato.
Estensioni	1. L'ordine dei giocatori non è disponibile o è vuoto. a. UC5 non può essere eseguito. Si verifica un errore di sistema da gestire.
Requisiti speciali	La gestione dei turni deve seguire l'ordine predeterminato dei giocatori. La logica deve distinguere correttamente tra giocatori che saltano turni fissi e giocatori bloccati da condizioni. I contatori di turno e giro devono essere aggiornati correttamente.
Elenco delle varianti tecnologiche e dei dati	Dati: lista dell'ordine dei giocatori. Metodi: passaggio turno, verifica e decremento dei turni saltati, logica di controllo del ciclo principale.
Frequenza delle ripetizioni	Una volta alla fine di ogni turno effettivo (non saltato) o una volta per ogni giocatore che deve saltare un turno fisso.
Varie	

3.2.6 UC6: Determinare il vincitore

Nome del caso d'uso	UC6: Determinare il vincitore
Portata	Partita in corso
Livello	Sistema
Attore primario	Sistema
Parti interessate e interessi	Tutti i giocatori: vogliono sapere chi ha vinto la partita e che il gioco è finito.
Precondizioni	La partita è in stato IN_CORSO. Un giocatore ha completato il suo turno (ha mosso e gli effetti, se presenti, sono stati applicati).
Garanzia di successo	Se un giocatore raggiunge esattamente la casella 63 alla fine del suo turno, viene dichiarato vincitore e lo stato della partita cambia a TERMINATA.
Scenario principale di successo	1. Il turno del giocatore corrente si conclude (dopo UC4). 2. Il sistema verifica la posizione finale del giocatore corrente. 3. Il sistema verifica se la posizione del giocatore corrente è esattamente 63. 4. Se la posizione è esattamente 63: a. Il sistema identifica il giocatore corrente come il vincitore. b. Il sistema cambia lo stato della partita a TERMINATA. c. Il sistema informa l'utente che un vincitore è stato determinato e quale giocatore ha vinto. d. Il ciclo di gioco principale termina. e. L'interfaccia utente ritorna al menu principale. 5. Se la posizione *non* è 63 e lo stato della partita non è già TERMINATA, la partita continua.
Estensioni	1. Più giocatori raggiungono la casella 63 contemporaneamente: a. la logica corrente verifica il vincitore dopo il turno *di ciascun* giocatore, garantendo che il primo a raggiungere 63 venga identificato. 2. La partita termina per un altro motivo. a. Il sistema gestisce l'errore; non viene dichiarato un vincitore secondo le regole del gioco.
Requisiti speciali	La condizione di vittoria deve essere l'arrivo *esattamente* sulla casella 63. L'identificazione del vincitore deve avvenire subito dopo che un giocatore completa il suo movimento e l'applicazione degli effetti.
Elenco delle varianti tecnologiche e dei dati	Dati: posizione del giocatore, stato della partita. Metodi: verifica vincitore, impostazione stato partita.

Frequenza delle ripetizioni	Almeno una volta alla fine di ogni turno giocato (non saltato), fino a che lo stato della partita non diventa TERMINATA
Varie	

3.3 Changelog completo dell'elaborazione

Questa elaborazione introduce significativi miglioramenti alle meccaniche di gioco e alla stabilità dell'interfaccia utente, allineandosi più fedelmente alle regole classiche del gioco dell'oca. La funzionalità di salvataggio è stata temporaneamente rimossa per riprogettazione nella prossima iterazione.

3.3.1 Modifiche principali

3.3.1.1 Configurazione partita

- Miglioramenti all'UI console:
  - Gestione avanzata degli errori di input, usando eccezioni per input illegali (ad esempio inserire un carattere e non un numero quando viene chiesto il numero dei giocatori);
  - Messaggi di errore più chiari con ripristino dello stato in caso di fallimento.
- Aggiunti controlli in ImpostazioniPartita per:
  - Nomi vuoti;
  - Valori fuori intervallo;
  - Assegnazione automatica di colori e tipi di giocatore (CPU/umano).

3.3.1.2 Gestione giri e turni

- Introduzione del concetto di giro:
  - Tracciamento del giroCorrente nella classe Partita.
  - Incremento automatico quando il turno ritorna al primo giocatore.
- Differenziazione tra:
  - Blocchi temporali (turniSaltati > 0, es. Pozzo = 3 turni).
  - Blocchi condizionali (turniSaltati < 0, es. Prigione = attesa di un "6" sul dado).

3.3.1.3 Meccanica di sblocco giocatore

- Nuova Funzionalità:
  - Metodo verificaELiberaGiocatoriBloccati(int[] valoreDadi) per verificare automaticamente se un lancio di dadi sblocca giocatori fermi in caselle speciali (es. un "6" libera dalla prigione).

3.3.1.4 Movimento ed effetti caselle

- Redesign del tabellone:
  - Assegnazione precisa di effetti a caselle specifiche (Ponte → casella 12, Labirinto → casella 35, ecc.).
  - Nuovi tipi di TipoEffettoCasella per una gestione più dettagliata delle regole.
- Separazione della logica degli effetti:
  - Casella.applicaEffetto(): gestisce effetti fissi (spostamenti, blocchi semplici).
  - PartitaController.gestisciEffettiSpeciali(): gestisce effetti dipendenti dai dadi (es. Raddoppio Movimento).

3.3.1.5 Interfaccia utente

- Visualizzazione migliorata:
  - Aggiunta di indicatori per "Giro X" e "Turno Y".
  - Dettagli sullo stato dei giocatori bloccati (es. "Giocatore: fermo per 2 turni (Pozzo)").
- Ottimizzazione del flusso di gioco:
  - Salti automatici dei turni bloccati con notifica.
  - Messaggi espliciti per i blocchi condizionali (es. "Lancia un 6 per liberarti!").

3.4 Contratti delle operazioni

Di seguito verranno introdotti i nuovi contratti, da aggiungere ai contratti esposti nell'iterazione 1 della fase di elaborazione:

Contratto CO13: confermaImpostazioni

Operazione:	GiocoController.confermaImpostazioni()
Riferimenti:	Fase finale di configurazione partita, inizio effettivo del gioco.
Pre-condizioni:	• Tutti i giocatori sono stati configurati (nome, tipo, colore). • L'interfaccia utente ha mostrato un riepilogo delle impostazioni e l'utente ha confermato ('S').
Post-condizioni:	• Un nuovo oggetto Partita è stato creato nel GiocoController, popolato con i giocatori configurati. • Il tabellone è stato inizializzato. Le posizioni iniziali dei giocatori sono state impostate a 0. • L'ordine di turno dei giocatori è stato determinato casualmente. Lo stato della partita è impostato su IN_CORSO. • L'interfaccia utente inizia a mostrare il tabellone e i giocatori, entrando nel ciclo di gioco principale.

Contratto CO14: mostraTabelloneEGiocatori

Operazione:	InterfacciaUtente.mostraTabelloneEGiocatori(Partita partita)
Riferimenti:	Ciclo principale del gioco.
Pre-condizioni:	Una partita è iniziata e il suo stato è IN_CORSO.

Post-condizioni:	• Il ciclo di gioco continua finché lo stato della partita non è TERMINATA.
	• Per ogni iterazione del ciclo:
	  ○ viene mostrato lo stato attuale del gioco (giocatori, posizioni, turno/giro);
	  ○ viene gestito il turno del giocatoreCorrente (lancio dadi, movimento, effetti, ecc.);
	  ○ (se la partita non è terminata e il giocatore non è bloccato da condizione) il turno passa al giocatore successivo.
	• Se la partita termina (vittoria o uscita), viene mostrato il vincitore (se presente) e l'interfaccia utente torna al menu principale dopo un input dell'utente.

Contratto CO15: passaAlProssimoGiocatore

Operazione:	Partita.passaAlProssimoGiocatore()
Riferimenti:	Flusso del turno di gioco.
Pre-condizioni:	Un giocatore ha completato il suo turno e non è bloccato da una condizione speciale (il suo turniSaltati è >= 0 dopo la verifica degli effetti o sblocchi). L'ordine dei giocatori nella partita non è vuoto.
Post-condizioni:	• Il giocatoreCorrente nella Partita viene aggiornato al giocatore successivo nell'ordine. • Il contatore turnoCorrente della Partita viene incrementato. • Se il passaggio del turno riporta all'inizio dell'ordine dei giocatori, il giroCorrente viene incrementato e turnoCorrente resettato a 1. • Se il nuovo giocatoreCorrente ha turniSaltati > 0, il suo contatore viene decrementato e il metodo continua a passare al giocatore successivo finché non ne trova uno con turniSaltati <= 0. • I giocatori con turniSaltati < 0 fermano questa catena di passaggi automatici.

Contratto CO16: gestisciTurnoUmano

Operazione:	InterfacciaUtente.gestisciTurnoUmano(Partita partita, Giocatore giocatore)
Riferimenti:	Sottociclo del turno per giocatore umano.
Pre-condizioni:	• È il turno del giocatoreCorrente che è di tipo Umano. • Il giocatore non deve saltare il turno a causa di un blocco fisso (turniSaltati > 0).

Post-condizioni:	• In base all'input dell'utente, il giocatore Umano può:
	  ○ Lanciare i dadi (attivando il flusso CO18, CO19, CO20)
	  ○ Tentare di uscire dalla partita (se conferma, lo stato della partita viene impostato su TERMINATA).
	• Se l'utente non lancia i dadi o annulla l'uscita, il menu del turno viene ripresentato.

Contratto CO17: gestisciTurnoComputer

Operazione:	InterfacciaUtente.gestisciTurnoComputer(Partita partita, Giocatore giocatore)
Riferimenti:	Sottociclo del turno per giocatore Computer.
Pre-condizioni:	• È il turno del giocatoreCorrente che è di tipo Computer. • Il giocatore non deve saltare il turno a causa di un blocco fisso (turniSaltati > 0).
Post-condizioni:	• Dopo l'input dell'utente per procedere, il giocatore Computer lancia automaticamente i dadi (attivando il flusso CO18, CO19, CO20). • Il suo turno si completa senza ulteriori interazioni da parte dell'utente fino al passaggio al giocatore successivo (a meno che non si blocchi condizionalmente).

Contratto CO18: tiraDadi

Operazione:	PartitaController.tiraDadi()
Riferimenti:	Azione di base nel turno di gioco.
Pre-condizioni:	È il turno di un giocatore (Umano o Computer) che non è bloccato per un numero fisso di turni (turniSaltati > 0).
Post-condizioni:	• Vengono generati due valori casuali tra 1 e 6, rappresentanti il risultato del lancio. • Il risultato viene mostrato all'utente (anche per la CPU). • Il sistema verifica la lista di tutti i giocatori per vedere se questo lancio sblocca qualcuno (inclusa la Prigione o Attesa Dado Specifico per il giocatore corrente), aggiornando il loro turniSaltati a 0 se le condizioni sono soddisfatte. • Se il giocatore corrente non è bloccato condizionalmente (turniSaltati >= 0) dopo questa verifica, il flusso prosegue con il movimento della pedina (CO19). • Se il giocatore corrente è ancora bloccato condizionalmente (turniSaltati < 0), il suo turno termina qui.

Contratto CO19: muoviPedina

Operazione:	PartitaController.muoviPedina(Giocatore giocatore, int passi)
Riferimenti:	Conseguenza del lancio di dadi.
Pre-condizioni:	• Un giocatore (Umano o Computer) ha lanciato i dadi (somma = passi) e non è bloccato da una condizione speciale (turniSaltati >= 0) dopo il lancio. • La posizione precedente del giocatore è nota.
Post-condizioni:	• La posizione del giocatore viene aggiornata aggiungendo passi. • Viene gestito il "rimbalzo" se la nuova posizione supera la casella 63. La casella di destinazione viene identificata. • La posizione aggiornata viene mostrata all'utente. • Il flusso prosegue con l'applicazione dell'effetto della casella di destinazione (CO20).

Contratto CO20: applicaEffettoCasella

Operazione:	PartitaController.applicaEffettoCasellaEsteso(Casella casella, Giocatore giocatore, int[] valoreDadi)
Riferimenti:	Conseguenza del movimento sulla casella di destinazione.
Pre-condizioni:	• Un giocatore è atterrato sulla casella di destinazione. • Lo stato della partita è IN_CORSO. • I valori specifici del lancio di dadi sono disponibili (valoreDadi).
Post-condizioni:	• Se la casella è la numero 63, lo stato della Partita è impostato su TERMINATA. Altrimenti, se la casella è speciale, il suo effetto base viene applicato (chiamando casella.applicaEffetto(giocatore)) modificando potenzialmente la posizione del giocatore, i turni da saltare o lo stato di rilancio. • Se l'effetto base della casella richiede un'ulteriore elaborazione basata sul risultato dei dadi (es. Raddoppia Movimento, Rilancia e Torna Indietro), la logica aggiuntiva viene eseguita, modificando ulteriormente lo stato del giocatore. • L'effetto applicato viene mostrato all'utente.

Contratto CO21: mostraVincitore

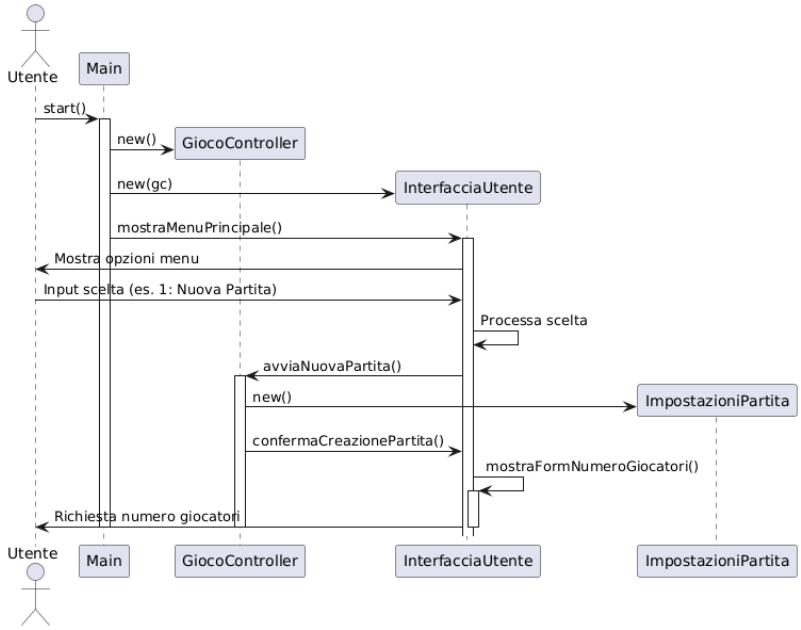
Operazione:	InterfacciaUtente.mostraVincitore(Giocatore giocatore)
Riferimenti:	Fine partita.
Pre-condizioni:	Lo stato della Partita è TERMINATA.
Post-condizioni:	• Viene mostrato un messaggio che annuncia il giocatore come vincitore. • L'interfaccia utente attende un input dell'utente (Invio) prima di tornare al menu principale.

3.5 Progettazione aggiornata

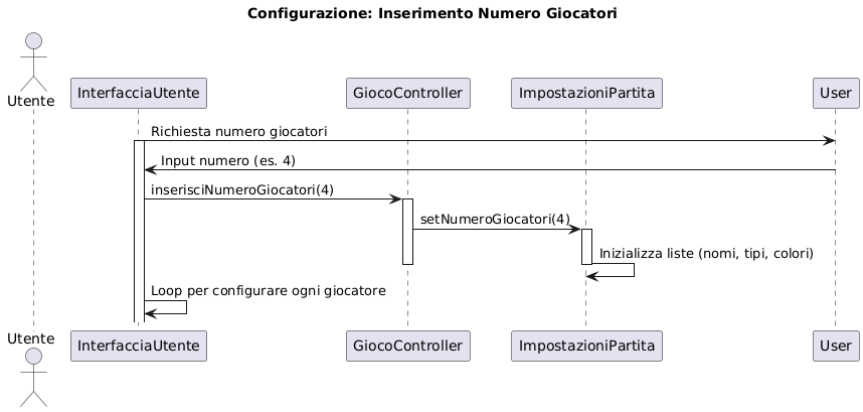
Conseguentemente al changelog, di seguito verranno mostrate tutte le versioni aggiornate e nuove dei diagrammi di sequenza.

3.5.1 Diagramma di sequenza

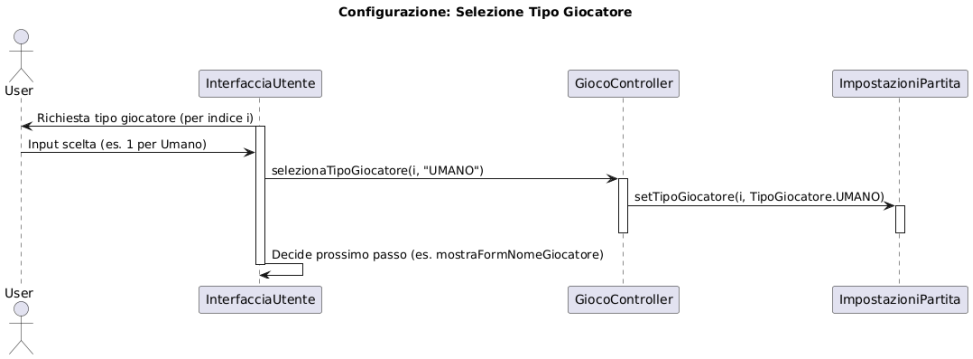
- Avvio partita e caricamento menù



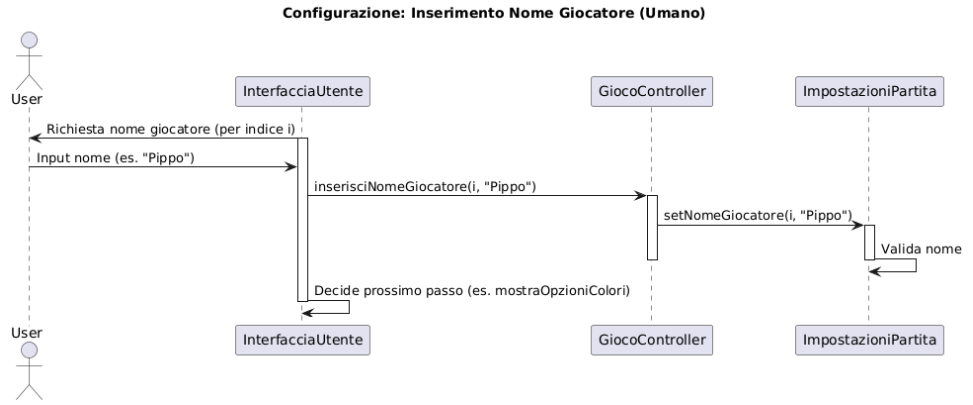
- Configurazione: inserimento giocatori



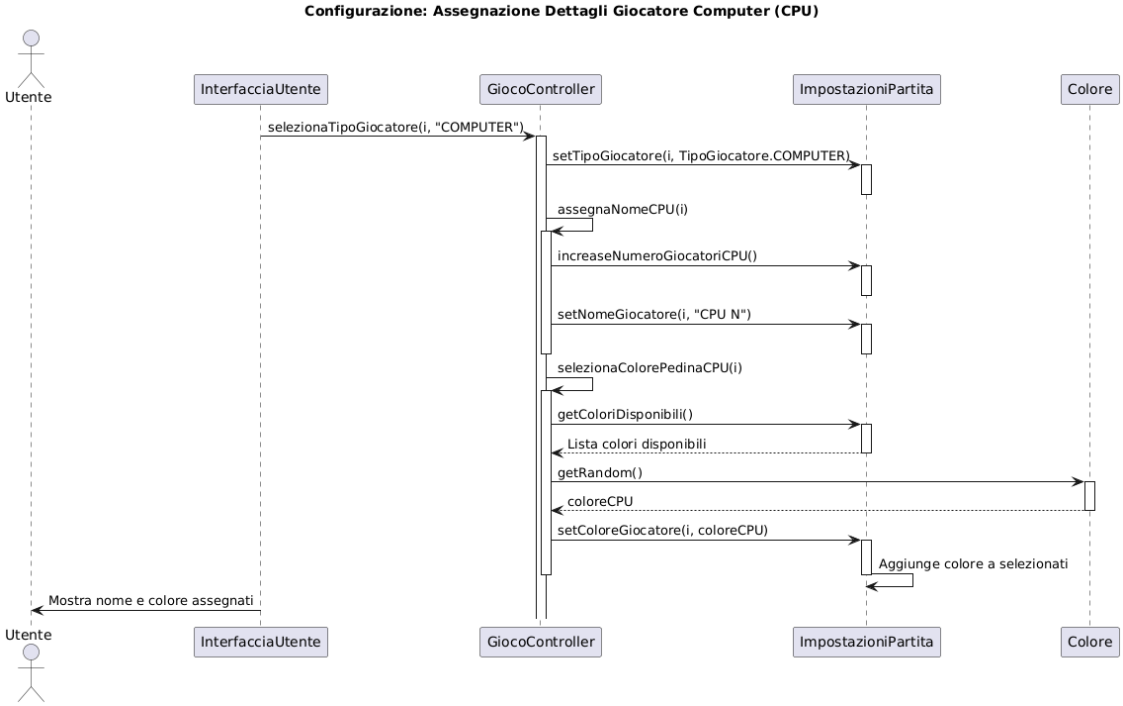
- Configurazione: selezione giocatore Umano/CPU



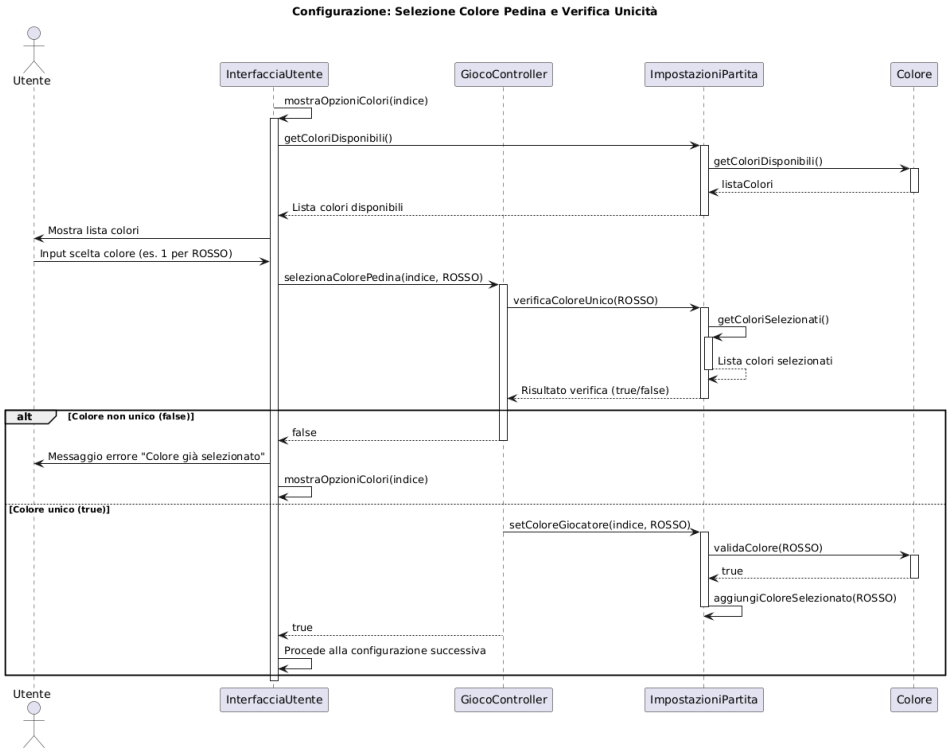
- Configurazione: inserimento nome giocatore Umano



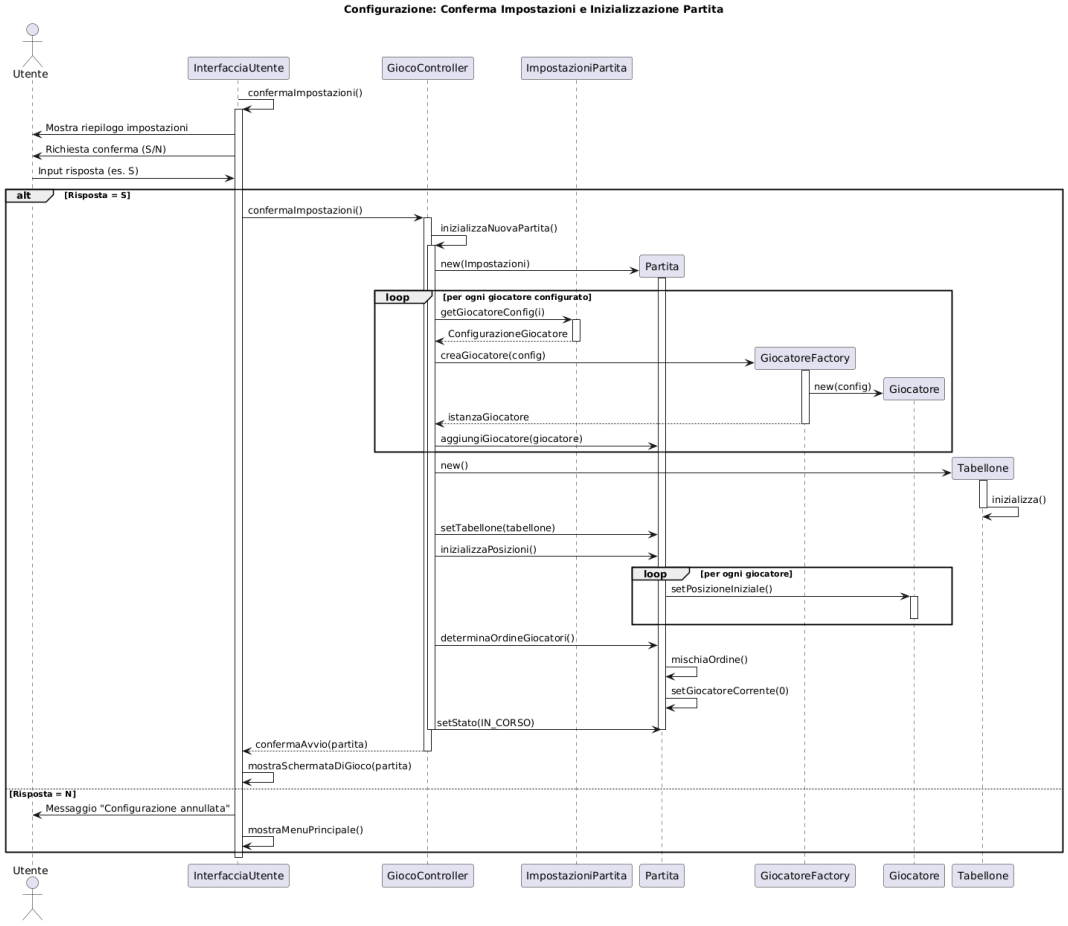
- Configurazione: assegnazione dettagli giocatore CPU



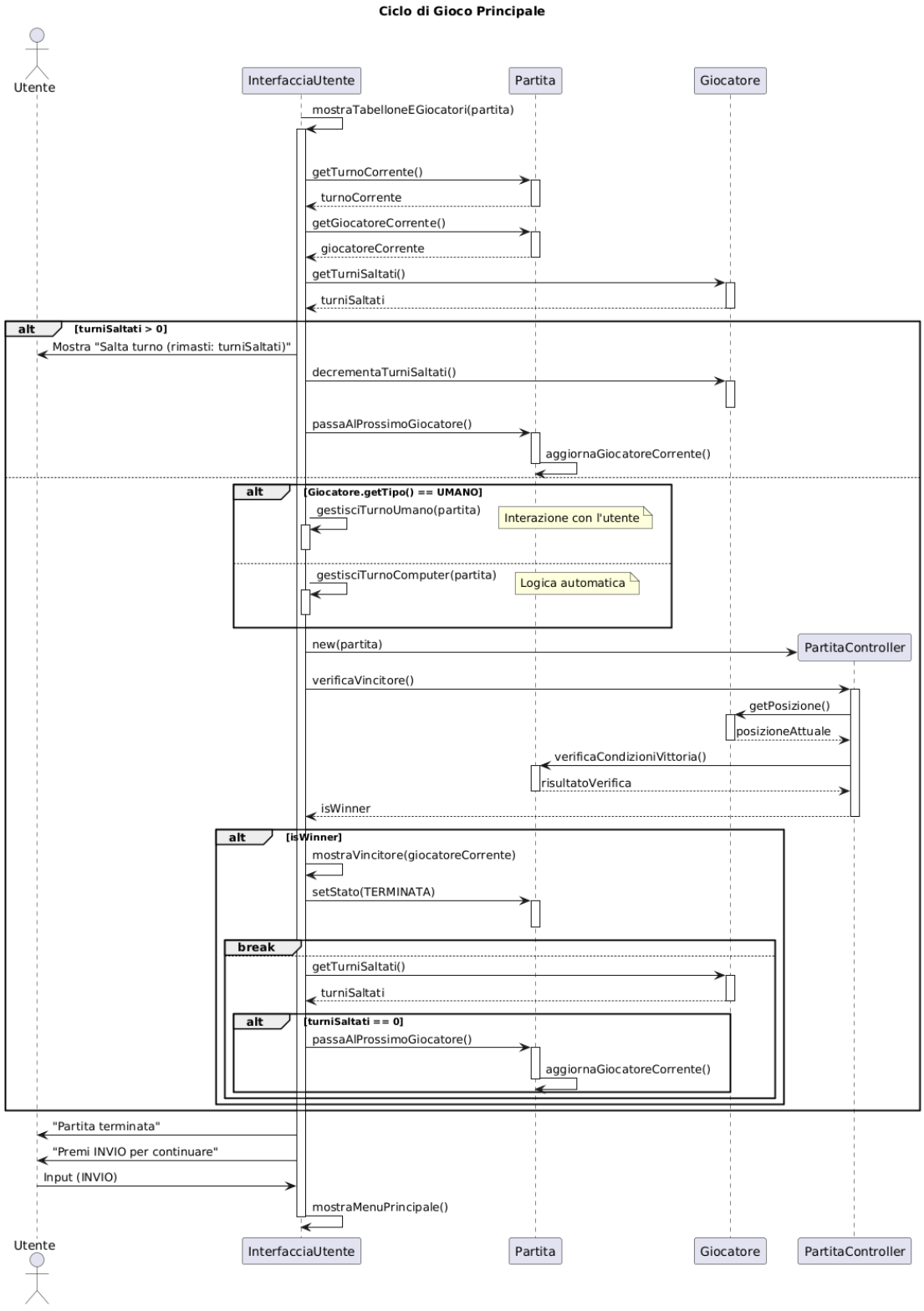
- Configurazione: selezione colore della pedina



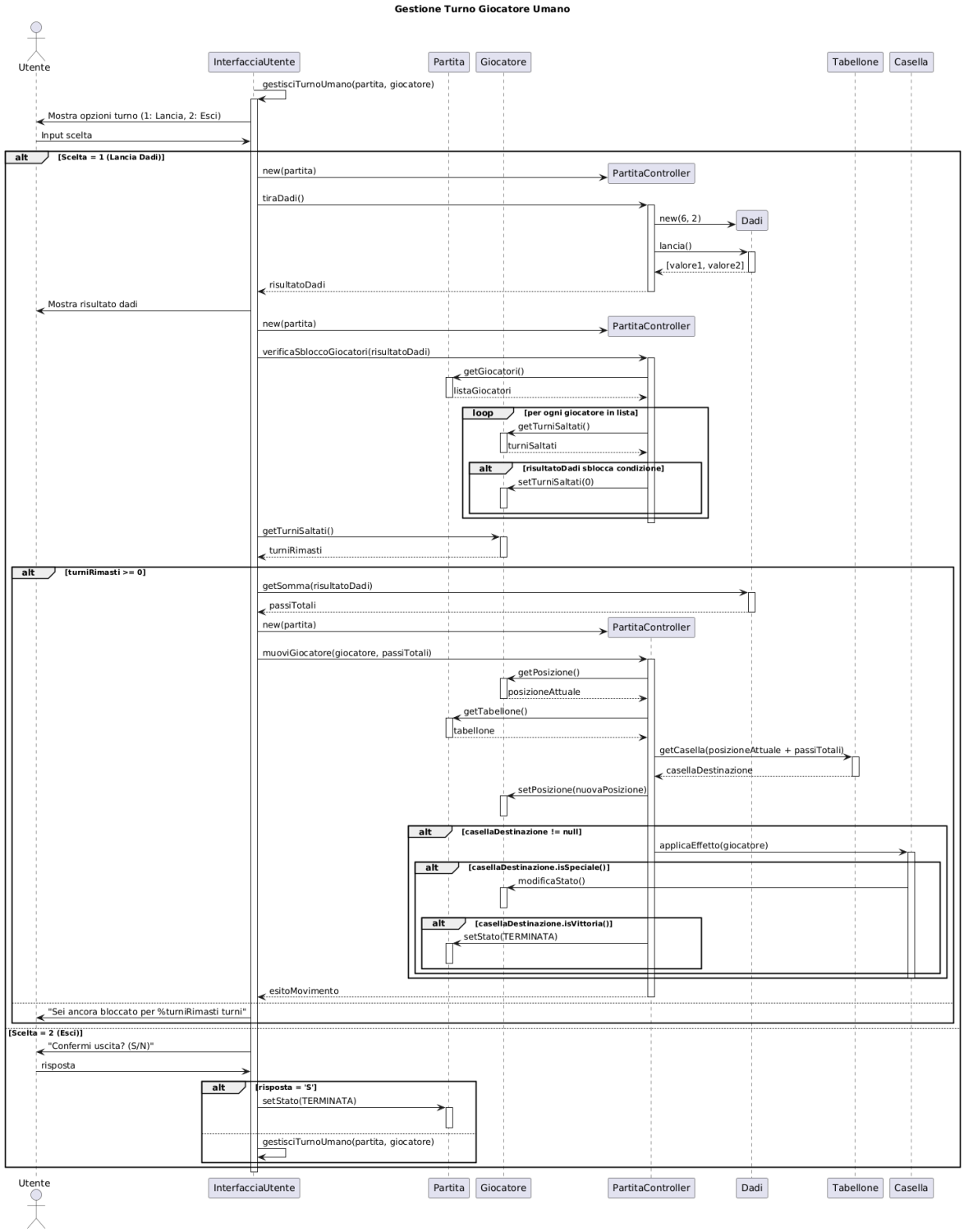
- Configurazione: inizializzazione partita



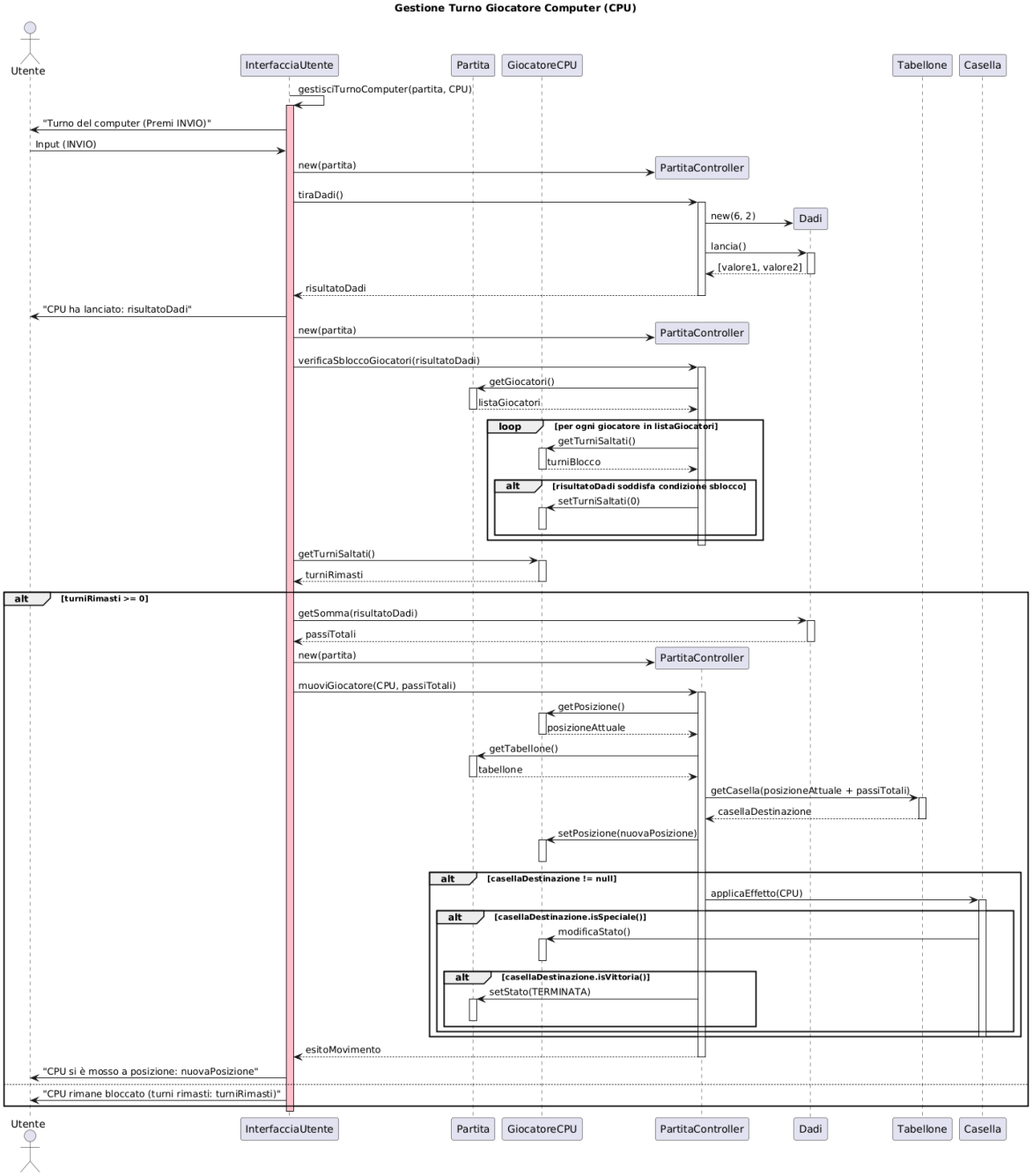
- Ciclo principale del gioco



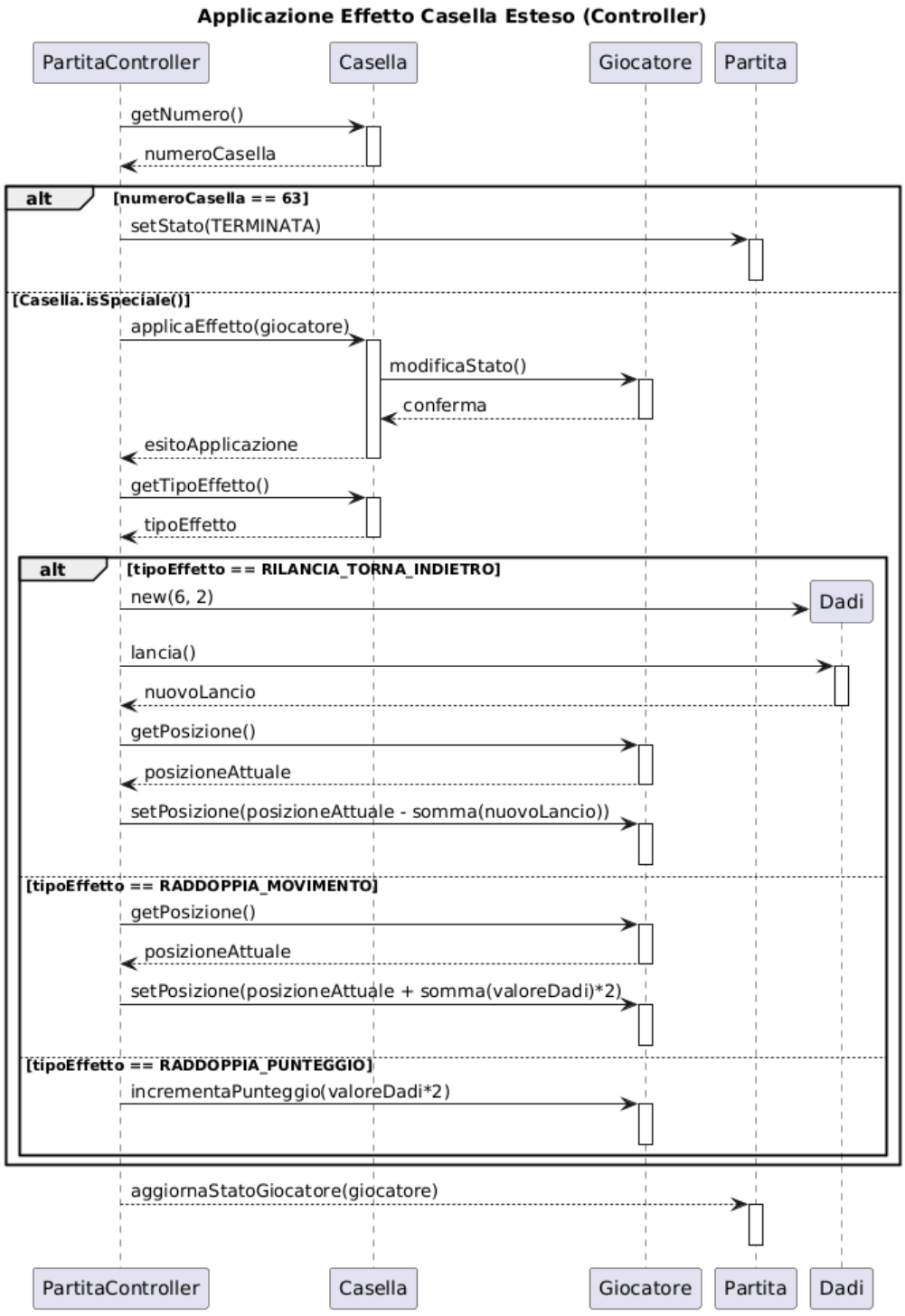
- Gestione turno del giocatore Umano



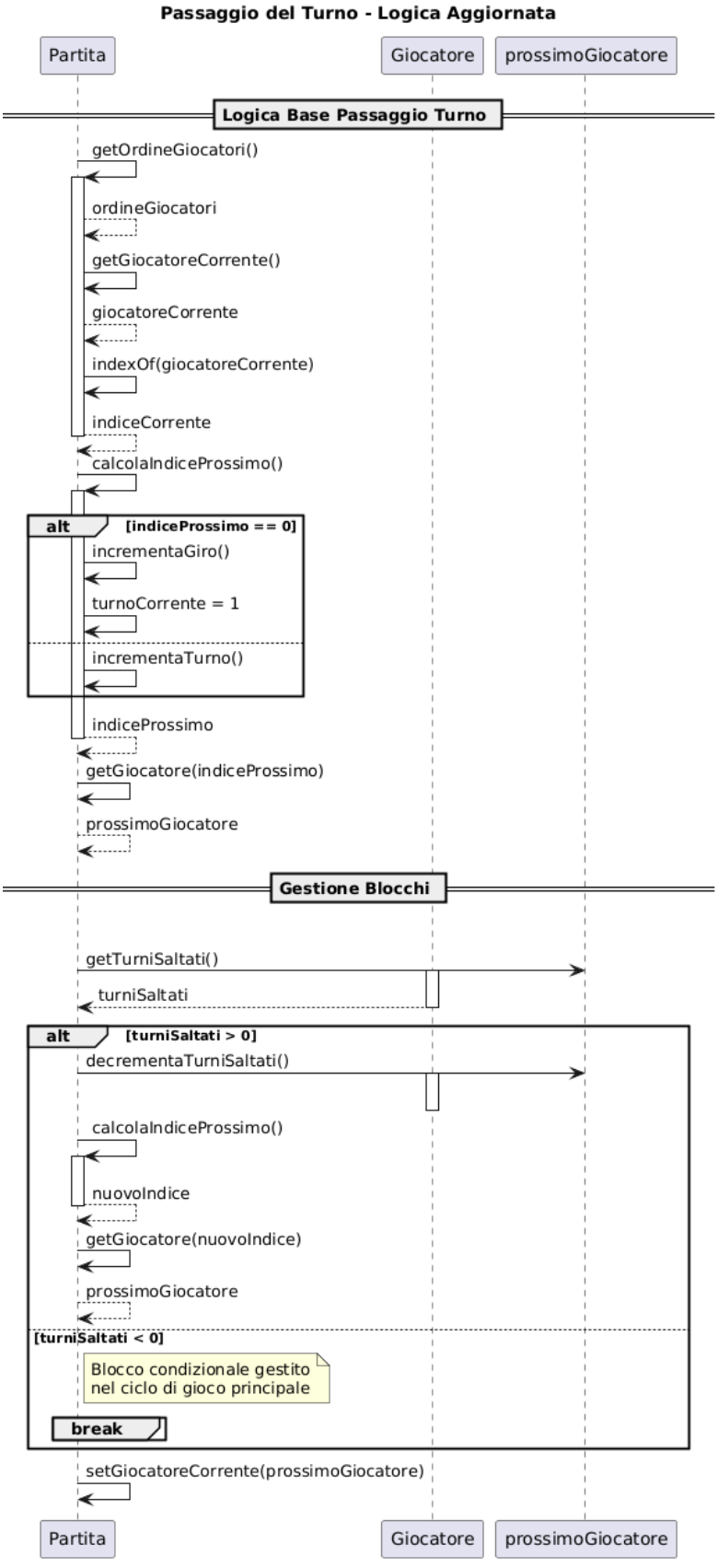
- Gestione turno del giocatore CPU



- Applicazione dell’effetto di una casella



- Gestione passaggio del turno



4 – Elaborazione – Iterazione 3

4.1 Introduzione

La terza iterazione si è concentrata sul completamento e l'affinamento di DigiGoose, portando l'applicazione da console a un'esperienza utente migliore. Gli obiettivi principali di questa fase sono stati l'introduzione della GUI, l'implementazione del sistema di persistenza per il salvataggio e caricamento delle partite, e l'irrobustimento della logica di gioco attraverso la risoluzione di bug e il perfezionamento delle meccaniche esistenti.

4.2 Implementazione dell'interfaccia utente con Swing

Per quanto riguarda la realizzazione dell'interfaccia grafica, è stata utilizzata la libreria Swing di Java. Questa evoluzione mira a migliorare significativamente l'esperienza dell'utente, rendendo il gioco più accessibile e intuitivo. Nei paragrafi seguenti, verranno analizzati i componenti chiave della nuova interfaccia grafica: la schermata di configurazione della partita, la rappresentazione interattiva del tabellone di gioco e le modalità con cui vengono comunicati visivamente gli effetti delle caselle speciali.

4.2.1 Configurazione della partita

La configurazione di una nuova partita è il primo passo interattivo che l'utente compie dopo aver selezionato "Nuova Partita" dal menu principale. Questa fase è gestita da un pannello dedicato, settingsPanel all'interno della classe InterfacciaUtente, che guida l'utente nella definizione dei parametri fondamentali del gioco.

4.2.1.1 Selezione del numero di giocatori e configurazione individuale

Al centro del pannello di configurazione si trova un JSpinner che permette all'utente di selezionare il numero di giocatori desiderato. La modifica del valore in questo spinner scatena un evento che invoca il metodo updatePlayerConfigPanelsContainer. Quest'ultimo è responsabile della creazione o rimozione dinamica di istanze della inner class PlayerConfigPanel. Per ogni giocatore selezionato, viene visualizzato un PlayerConfigPanel dedicato, organizzato verticalmente all'interno di un JPanel con BoxLayout, a sua volta contenuto in un JScrollPane per gestire un numero variabile di giocatori senza compromettere il layout.

Ogni PlayerConfigPanel incapsula gli elementi UI necessari per configurare un singolo giocatore:

1. **Tipo Giocatore**: Una JComboBox permette di scegliere se il giocatore sarà UMANO o COMPUTER. La selezione di COMPUTER disabilita il campo nome e lo precompila con un identificativo standard (es. "CPU 1").
2. **Nome Giocatore**: Un JTextField per inserire il nome del giocatore. Questo campo è attivo solo se il tipo selezionato è UMANO.
3. **Colore Pedina**: Una JComboBox elenca tutti i colori disponibili definiti nell'enum Colore. Viene preselezionato un colore diverso per ogni pannello per facilitare la scelta di colori unici, sebbene la validazione finale avvenga al momento della conferma.

4.2.1.2 Raccolta, validazione e conferma delle impostazioni

Una volta che l'utente ha definito le impostazioni per tutti i giocatori, preme il pulsante "Conferma Impostazioni". Questa azione scatena il metodo raccogliEConfermaImpostazioni in InterfacciaUtente.

Vengono eseguite delle validazioni cruciali:

- Il nome per i giocatori umani non può essere vuoto.
- Ogni giocatore deve aver selezionato un colore.
- Tutti i colori selezionati devono essere unici tra i giocatori.

Se una di queste validazioni fallisce, viene mostrato un messaggio d'errore specifico, e il processo di conferma si interrompe, permettendo all'utente di correggere gli input.

Se tutte le validazioni hanno successo, InterfacciaUtente invoca i metodi appropriati del GiocoController. Infine, viene chiamato giocoController.confermaImpostazioni(), che a sua volta invoca inizializzaNuovaPartita() per creare l'oggetto Partita con i dati raccolti.

Il design adottato permette una chiara separazione delle responsabilità: la InterfacciaUtente si occupa della presentazione e della raccolta dell'input, il GiocoController orchestra il processo di configurazione e valida i dati a un livello più alto, e ImpostazioniPartita agisce come un data-holder temporaneo prima della creazione effettiva dell'oggetto Partita.

4.2.2 Tabellone di gioco

Una volta completata la configurazione e avviata la partita, l'interfaccia utente presenta il tabellone di gioco. La sua visualizzazione e interazione sono gestite dalla classe GameBoardPanel, un JPanel personalizzato progettato specificamente per disegnare dinamicamente lo stato del gioco.

4.2.2.1 Struttura e layout del tabellone

Il GameBoardPanel è responsabile della rappresentazione grafica delle 64 caselle del gioco. Al momento della sua inizializzazione, o quando le dimensioni del pannello cambiano, il metodo calculateSquarePositions viene invocato. Questo metodo calcola le coordinate per ogni casella, disponendole in una griglia. La dimensione di ogni casella e il numero di caselle per riga sono calcolati dinamicamente per adattarsi allo spazio disponibile nel pannello contenitore, garantendo una visualizzazione ottimale su diverse risoluzioni, pur mantenendo una dimensione minima preferita per la leggibilità.

4.2.2.2 Rendering delle caselle e delle informazioni

Il cuore del rendering avviene nel metodo paintComponent(Graphics g), che viene sovrascritto per disegnare tutti gli elementi del tabellone. Ogni casella è un rettangolo, dove il colore di sfondo varia per distinguere le caselle normali (bianco), la casella di partenza (grigio chiaro), la casella di arrivo (arancione) e le caselle speciali (giallo chiaro).

4.2.2.3 Visualizzazione delle pedine dei giocatori

Le pedine dei giocatori sono rappresentate da ovali colorati disegnati sopra le caselle corrispondenti alla loro posizione attuale. Il GameBoardPanel itera sulla lista dei giocatori della partita corrente:

1. Per ogni giocatore, recupera la sua posizione e il colore della sua pedina. I colori definiti nell'enum it.digigoose.model.Colore sono mappati a istanze di java.awt.Color attraverso una HashMap per il rendering.
2. Per gestire la sovrapposizione di più pedine sulla stessa casella, viene calcolato un offset. Le pedine vengono disposte in una piccola griglia all'interno della casella (fino a 3 pedine per riga virtuale all'interno della casella), evitando che si coprano completamente. Una HashMap tiene traccia di quante pedine sono già state disegnate su una specifica casella per calcolare l'offset corretto.
3. Su ogni pedina viene disegnata l'iniziale del nome del giocatore, con un colore di testo (bianco o nero) scelto per garantire il contrasto con il colore della pedina.

4.2.2.4 Aggiornamento del board durante la partita

Ogni volta che lo stato della partita cambia (es. un giocatore si muove), il metodo repaint() viene chiamato sul GameBoardPanel. Questo forza la riesecuzione di paintComponent, assicurando che il tabellone rifletta sempre lo stato più recente del gioco, incluse le posizioni delle pedine e le informazioni visualizzate.

4.2.3 Visualizzazione effetti caselle speciali

Per aiutare i giocatori a capire cosa succede quando si finisce su una casella speciale, il gioco offre informazioni chiare in due modi: attraverso piccoli messaggi che appaiono quando si passa il mouse sopra una casella e tramite aggiornamenti testuali durante la partita.Quando il giocatore muove il mouse sul tabellone, il sistema è attento a dove si trova il cursore. Se il mouse si ferma per un secondo su una casella speciale (o su quella di partenza o di arrivo), appare una piccola finestra con una spiegazione. Questo piccolo ritardo evita che i messaggi appaiano e scompaiano troppo velocemente se si muove il mouse rapidamente. Se il mouse si sposta via dalla casella o esce dal tabellne, la finestra informativa scompare.Dentro il tooltip è presente come titolo il nome della casella speciale preso dall'enum e una piccola descrizione dell'effetto. In basso della finestra, rimane l'output testuale della console, potendo quindi seguire ogni passaggio del flusso di gioco.

4.3 Sistema di salvataggio e caricamento partita con JSON

Per migliorare l'esperienza di gioco e permettere agli utenti di interrompere e riprendere le proprie sessioni, è stato implementato un sistema di persistenza dei dati. Questa funzionalità consente di salvare lo stato corrente di una partita in un file e di ricaricarlo successivamente. È stato scelto il formato JSON per la sua leggibilità e la facile interoperabilità, avvalendosi della libreria org.json, scaricata tramite il seguente link su maven.org e incluso nella cartella lib del progetto:

https://repo1.maven.org/maven2/org/json/json/20250517/

Il sistema gestisce un singolo slot di salvataggio, sovrascrivendo automaticamente il file esistente a ogni nuovo salvataggio.

4.3.1 Creazione del file JSON per lo stato del gioco

Il processo di salvataggio è orchestrato dalla classe GestoreSalvataggio, specificamente attraverso il metodo salvaPartita(Partita partita). Quando l'utente sceglie di salvare la partita (tipicamente prima di tornare al menu principale da una partita in corso), questo metodo viene invocato. Le informazioni cruciali della partita vengono convertite in una struttura JSON.

Viene creato un oggetto JSON principale che rappresenta la partita. Al suo interno, vengono memorizzati attributi fondamentali come l'ID univoco della partita, la data di creazione, il turno e il giro correnti, e lo stato attuale del gioco (es. IN_CORSO).

Una parte significativa dell'oggetto JSON è dedicata alla lista dei giocatori. Per ogni giocatore, viene creato un sotto-oggetto JSON che include il suo ID, nome, tipo (umano o CPU), il numero di turni che deve eventualmente saltare e lo stato della sua richiesta di rilancio. Anche i dettagli della pedina associata a ciascun giocatore, come il colore e la posizione corrente sul tabellone, sono serializzati all'interno dell'oggetto del rispettivo giocatore.

L'ordine di gioco dei giocatori, che potrebbe essere diverso dall'ordine di creazione a causa del mescolamento iniziale, viene preservato memorizzando una lista degli ID dei giocatori nell'ordine corretto. Viene inoltre salvato l'ID del giocatore il cui turno è attualmente in corso.

Una volta che l'intero stato della partita è stato tradotto in questa struttura JSON gerarchica, l'oggetto JSON viene scritto su un file denominato digigoose_save.json all'interno di una sottodirectory save/. Se un file di salvataggio esiste già, viene sovrascritto.

4.3.2 Gestione del caricamento della partita

La funzionalità di caricamento permette di ripristinare una partita precedentemente salvata. All'avvio dell'applicazione, InterfacciaUtente verifica, tramite gestoreSalvataggio.esisteSalvataggio(), la presenza del file digigoose_save.json. Se il file esiste, il pulsante "Carica Partita" nel menu principale viene reso visibile e attivo.

Selezionando "Carica Partita", viene invocato il metodo caricaPartita() di GestoreSalvataggio. Questo metodo legge il contenuto del file JSON. Il testo JSON viene parsato e riconvertito in un oggetto.

Da questo oggetto principale, vengono estratte le varie informazioni per ricostruire l'oggetto Partita e i suoi componenti. Si ricreano le istanze dei Giocatore con i loro attributi (nome, tipo, colore pedina, posizione, turni saltati, richiesta rilancio), assicurandosi di ripristinare anche i loro ID originali. La ricostruzione dell'ordine dei giocatori e l'identificazione del giocatore corrente avvengono utilizzando gli ID salvati, mappandoli ai giocatori appena ricreati.

Una volta che l'oggetto Partita è stato completamente ripristinato in memoria con i dati del file di salvataggio, viene passato al GiocoController e al PartitaController. L'interfaccia utente passa quindi direttamente alla schermata di gioco (gamePlayPanel), inizializzando la logica di gioco (iniziaLogicaDiGioco()) con lo stato caricato. Viene visualizzato un messaggio di conferma del caricamento.

Se durante il processo di caricamento si verificano errori, ad esempio a causa di un file corrotto o mancante (nonostante il controllo iniziale), vengono gestite le eccezioni (IOException) e l'utente viene informato tramite un messaggio di errore, tipicamente rimanendo nel menu principale.

4.4 Irrobustimento della logica di gioco e correzioni bug

Oltre all'introduzione di nuove funzionalità come l'interfaccia grafica e il sistema di salvataggio, una parte significativa del lavoro in questa iterazione finale è stata dedicata al miglioramento della stabilità e della correttezza della logica di gioco. Questo ha comportato l'analisi e la risoluzione di alcuni bug emersi durante i test e il raffinamento di meccaniche di gioco complesse per garantire un'esperienza fluida e aderente alle regole.

4.4.1 Gestione bug rilancio dei dadi

Una delle meccaniche fondamentali del gioco dell'oca è la possibilità, offerta da alcune caselle speciali, di far rilanciare i dadi al giocatore. Durante le fasi di test precedenti, era emersa una problematica legata alla gestione di questa situazione, in particolare quando il giocatore che otteneva il diritto al rilancio era un computer (CPU) o quando il rilancio si verificava in contesti di effetti a catena.

La problematica principale risiedeva nel flusso di controllo del turno. In alcuni scenari, dopo che un giocatore (specialmente una CPU) aveva ottenuto un rilancio, il sistema poteva passare erroneamente il turno al giocatore successivo prima che il rilancio venisse effettivamente eseguito, oppure il pulsante per il rilancio per un giocatore umano non si riattivava correttamente.

La correzione ha implicato una revisione del metodo concludiTurnoEAvanza in InterfacciaUtente. Ora, prima di passare il turno, si verifica esplicitamente se il giocatoreCorrente ha l'attributo richiedeRelancio impostato a true.

- Se un giocatore umano ha diritto al rilancio, il pulsante rollDiceButton viene riabilitato, e il turno non passa. L'utente vedrà un messaggio che lo invita a lanciare di nuovo.
- Se un giocatore CPU ha diritto al rilancio, viene avviato un nuovo Timer (computerReRollTimer) che, dopo un breve intervallo per simulare la riflessione, invoca nuovamente gestisciTurnoGiocatore. Anche in questo caso, il turno non passa al giocatore successivo finché il rilancio non è stato completato.
  L'attributo richiedeRelancio del giocatore viene resettato a false solo dopo che il rilancio è stato effettivamente gestito, assicurando che l'azione venga eseguita una sola volta come previsto.

Inoltre, è stata migliorata la gestione della logica di gioco quando un effetto speciale come RILANCIA_TORNA_INDIETRO viene attivato. In questo caso, l'interfaccia utente ora gestisce un secondo lancio di dadi dedicato (invocando partitaController.tiraDadi()) il cui risultato determina di quanti passi il giocatore deve tornare indietro. Questo isola il "rilancio per tornare indietro" dal normale flusso di un "rilancio per avanzare", prevenendo confusioni e garantendo che l'effetto venga applicato correttamente e in modo trasparente per l'utente, con messaggi appropriati nell'area di testo.

4.4.2 Gestione turni giocatori bloccati

Diverse caselle speciali nel Gioco dell'Oca impongono restrizioni ai giocatori, bloccandoli per un certo numero di turni o fino al verificarsi di una specifica condizione (ad esempio, ottenere un particolare risultato con i dadi). La gestione di questi stati di "blocco" è stata significativamente irrobustita per coprire tutti gli scenari e garantire che i giocatori vengano sbloccati correttamente.

La logica di gestione dei giocatori bloccati è centralizzata principalmente nel metodo prepareNextTurn di InterfacciaUtente e nel metodo verificaELiberaGiocatoriBloccati di PartitaController.
Quando è il turno di un giocatore (prepareNextTurn):

1. Si controlla l'attributo turniSaltati del giocatore corrente.
    - Se turniSaltati > 0: il giocatore deve saltare un numero fisso di turni. Viene visualizzato un messaggio, turniSaltati viene decrementato, e il turno passa immediatamente al giocatore successivo (tramite una chiamata ricorsiva a prepareNextTurn dopo un breve Timer per dare tempo all'utente di leggere il messaggio).
    - Se turniSaltati < 0: il giocatore è bloccato da una condizione speciale (es. in attesa di un 6 per uscire dalla Prigione, o di un 3/6 sulla casella 26, o di un 4/5 sulla casella 53). Viene visualizzato un messaggio specifico che descrive la condizione di blocco. Il giocatore tenterà comunque di lanciare i dadi.
    - Se turniSaltati == 0: il giocatore è libero di agire.

Il metodo gestisciTurnoGiocatore in InterfacciaUtente coordina il lancio dei dadi. Subito dopo il lancio, viene invocato partitaController.verificaELiberaGiocatoriBloccati(currentDiceRoll). Questo metodo analizza il risultato dei dadi appena lanciati:

- Controlla se nei dadi è presente un 6. Se sì, tutti i giocatori che erano bloccati con turniSaltati == -1 (effetto "ATTENDI_DADO" generico, che prima non era chiaramente implementato e ora è implicitamente coperto da caselle specifiche come la Prigione se si volesse generalizzare, anche se nel codice attuale -1 è usato per la casella 19 originale, ora FERMA_UN_TURNO) vengono sbloccati (il loro turniSaltati è posto a 0).
- Se il giocatore *corrente* era bloccato in Prigione (turniSaltati == -2, ora associato a FERMA_DUE_TURNI o specifici turniSaltati positivi dalla casella Prigione 52) e ha lanciato un 6, viene sbloccato.
- Se il giocatore *corrente* era sulla casella 26 (turniSaltati == -3) e ha lanciato un 3 o un 6, viene sbloccato.
- Se il giocatore *corrente* era sulla casella 53 (turniSaltati == -4) e ha lanciato un 4 o un 5, viene sbloccato.

Se, dopo il lancio dei dadi e la verifica, il giocatore corrente risulta ancora bloccato (cioè turniSaltati < 0), il suo movimento viene impedito, un messaggio informa l'utente che non ha ottenuto il risultato necessario, e il turno passa al giocatore successivo. Se invece viene sbloccato o non era bloccato, procede con il movimento.

È stata inoltre chiarita la comunicazione all'utente dello stato di blocco. La funzione getBlockMessage in InterfacciaUtente fornisce stringhe di testo appropriate per ogni tipo di blocco condizionale, che vengono visualizzate nell'area messaggi.

Questo approccio più strutturato assicura che le condizioni di blocco e sblocco siano gestite correttamente, che il flusso del gioco non si interrompa inaspettatamente e che l'utente riceva un feedback chiaro sullo stato dei giocatori bloccati e sui requisiti per il loro sblocco.