

# Iterazione 2

## 1.0 Introduzione

La seconda iterazione si concentrerà sui seguenti aspetti:

- Implementazione di estensioni dei casi d'uso già implementati.
- Implementazione di nuovi casi d'uso.
- Utilizzo di Pattern GoF per raffinare il modello di dominio.
- Refactoring del codice e testing.

In particolare, in questa seconda iterazione andremo ad implementare gli scenari principali di successo dei seguenti casi d'uso:

- **UC4:** Inserimento tipologia mazzo
- **UC5:** Creazione tabellone
- **UC6:** Gestisci eliminazione

Nel corso dell'iterazione, sono inoltrate state considerate, modellate e implementate le regole di dominio: R1, R2, R5, R8, R9.

### 1.1 Aggiornamento dei casi d'uso

Come per la prima iterazione, durante questa fase sono stati specificati e rivisti dei dettagli dei casi d'uso implementati. Per visionare le modifiche apportate ai suddetti, si consulti il documento “Modello dei casi d'uso” riportato nella cartella in versione aggiornata per questa iterazione. In particolare, è stato rivisto e aggiornato a dovere il caso d'uso UC8. Se verranno apportate ulteriori modifiche anche a documenti precedenti, queste saranno tramite cronologia. Inoltre, sono state raffinate le regole di dominio del sistema, si faccia riferimento al documento nella directory corrente per visionare tali modifiche.

## 2.0 Fase di Analisi

Nella fase di Analisi della seconda iterazione, sono state modellate due nuove classi concettuali per permettere l'introduzione di alcuni casi d'uso. Inoltre è stato modellata l'Enumeration “Gioco” dei giochi supportati dal sistema.

- **Tabellone:** rappresenta il tabellone di un torneo, che viene aggiornato progressivamente con l'eliminazione dei concorrenti.
- **Partita:** rappresenta la partita disputata tra due giocatori all'interno di un torneo, risulta utile per aggiornare il tabellone.
- **Gioco:** rappresenta i giochi supportati dal sistema, per semplificare e rendere scalabile la loro gestione ed utilizzo.

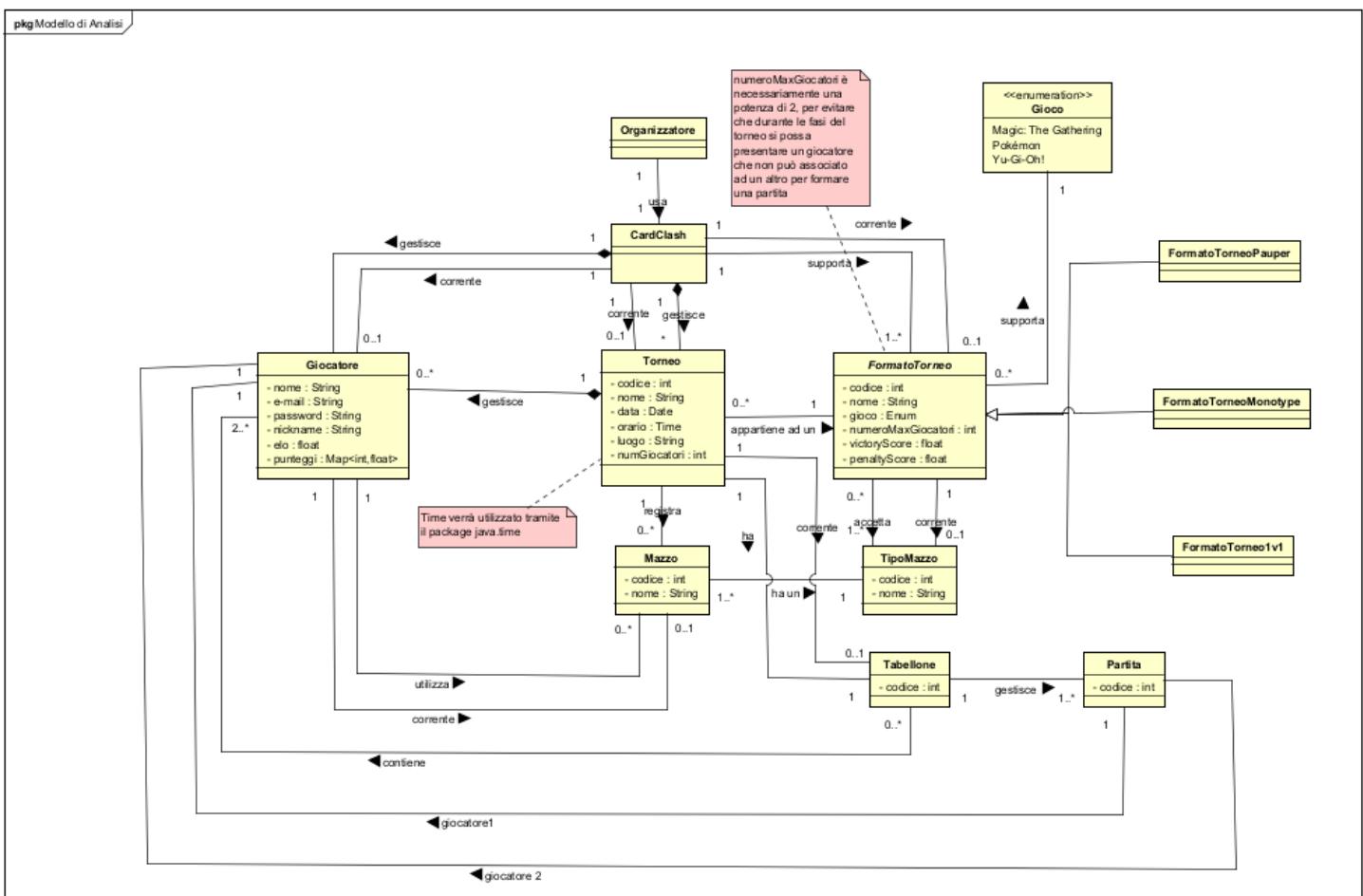
Inoltre, sono stati aggiornati alcuni parametri di classi concettuali già definite in precedenza:

- Aggiunto l'attributo numeroMaxGiocatori su FormatoTorneo, per permettere una corretta implementazione dell'iscrizione ad un torneo, non superando il limite di giocatori consentito dal formato a cui appartiene.
- Modificato l'attributo “gioco” su FormatoTorneo. Questa scelta deriva da numerose riflessioni fatte sull'intero funzionamento dell'applicativo, anche in ottica di ottimizzazione del codice per il caricamento dei formati stessi supportati dal sistema. In particolare, utilizzando una Enumeration per memorizzare i giochi del sistema, evitiamo in primis di creare una classe per memorizzare unicamente una sola stringa contenente il nome, ed in secondo luogo si va a rendere più sicuro e scalabile il processo di caricamento (e di inserimento/creazione, vedi UC9) dei formati di torneo del sistema. Chiaramente, a seguito di questa scelta è stata effettuato un profondo refactoring del codice, si vedano le modifiche nella directory apposita.
- Aggiunti gli attributi victoryScore e penaltyScore su FormatoTorneo, per implementare la gestione dei punteggi dei giocatori (e le eventuali future estensioni circa le penalità)
- Aggiunta l'associazione formatoCorrente su CardClash, per gestire l'inserimento di nuove tipologie di mazzo (UC4)
- Aggiunto l'attributo “elo” su Giocatore, per permettere l'aggiornamento dell'ELO del giocatore in base ai punti guadagnati alla fine del torneo. Aggiunta la mappa “punteggi” per consentire, per ogni giocatore, di tenere traccia del punteggio ottenuto in uno specifico torneo (risultetà utile per costruire le classifiche, si veda UC7)
- Aggiunto l'attributo numGiocatori su Torneo, per consentire di controllare, al momento della creazione del tabellone, se il numero di giocatori è correttamente una potenza di 2 (estensione 3.a, si riflette anche sull'implementazione del metodo

isAperto lato codice). L'inserimento di questo attributo prevede una leggera modifica del'SD "confermalscrizione" di UC3, che verrà riportato in seguito.

- Aggiunta l'associazione tipoMazzoCorrente tra FormatoTorneo e TipoMazzo per gestire correttamente gli inserimenti.
  - Si è deciso di utilizzare il pattern GoF Template Method: la classe FormatoTorneo è stata trasformata in una classe astratta da cui derivano tre implementazioni: FormatoTorneoPauper, FormatoTorneoMonotype e FormatoTorneo1v1. Questa struttura permette di implementare metodi specifici per ciascun tipo di torneo, garantendo flessibilità e personalizzazione in base al formato selezionato.

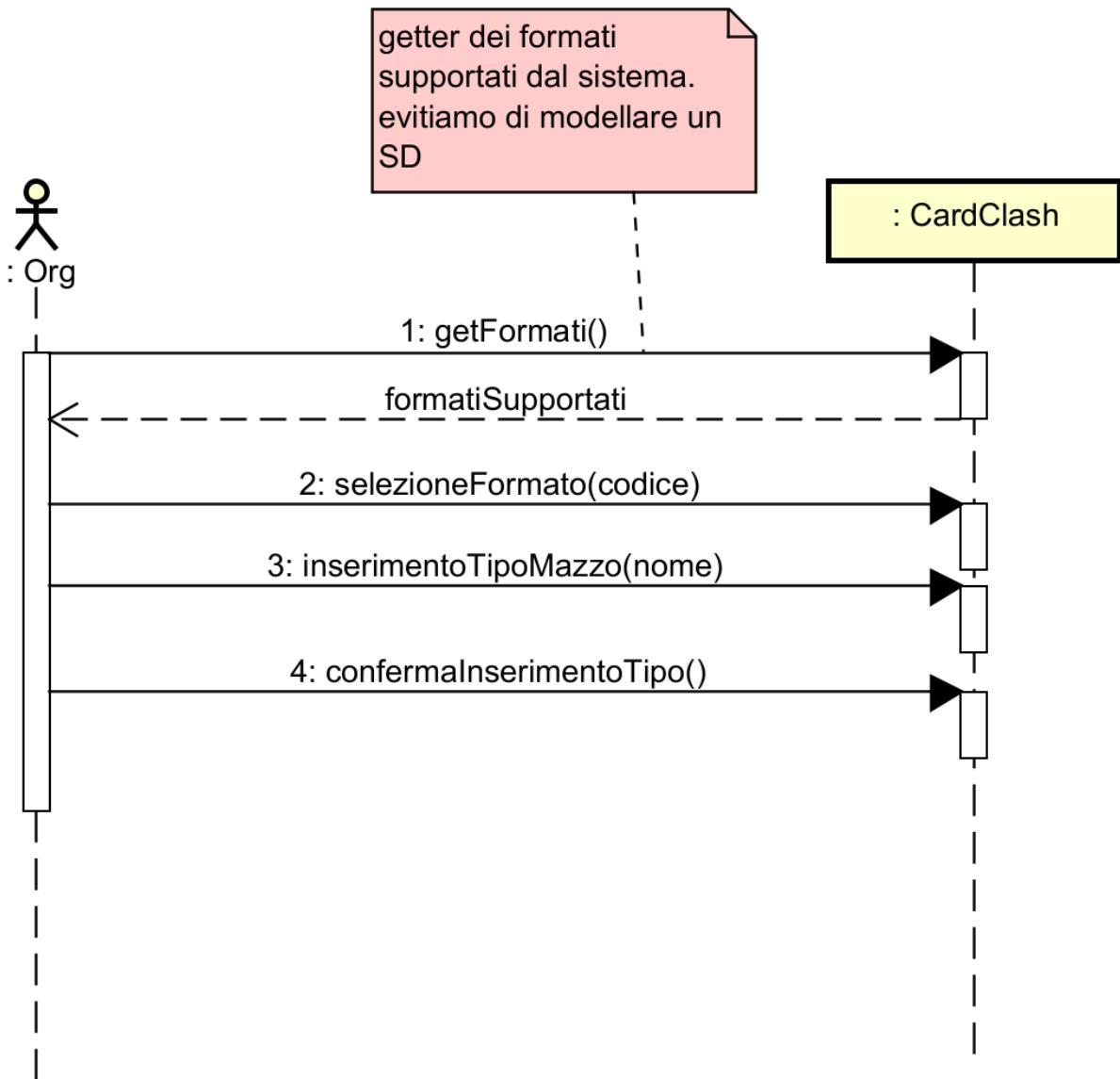
Si riporta di seguito il Modello di Dominio aggiornato:



## 2.1 Diagramma di sequenza di sistema

- **UC4:** Inserimento tipologia mazzo

Il seguente Diagramma di Sequenza di Sistema (SSD) illustra il corso di eventi I/O tra l'organizzatore e il sistema CardClash.



### 2.1.1 Contratti delle operazioni

I seguenti Contratti descrivono le principali operazioni eseguite dal sistema sulla base degli eventi individuati nell'SSD.

## Contratto1 UC4: Selezione del Formato

- **Operazione:** selezioneFormato(codice)
- **Riferimenti: Caso d'uso:** Inserimento tipologia Mazzo
- **Pre-condizione:** L'operazione di get dei formati supportati è andata a buon fine, e l'amministratore risulta essere autenticato.
- **Post-condizione:**
  - Viene recuperato il formato di torneo selezionato f
  - Il formato selezionato f diventa corrente per CardClash

## Contratto2 UC4: Inserimento del tipo del mazzo

- **Operazione:** inserimentoTipoMazzo(nome)
- **Riferimenti: Caso d'uso:** Inserimento tipologia Mazzo
- **Pre-condizione:**
  - È stato selezionato un formato supportato f
  - f è corrente per CardClash
- **Post-condizione:**
  - Viene creata una istanza TipoMazzo tm
  - Gli attributi di tm vengono inizializzati
  - tm diventa corrente per f

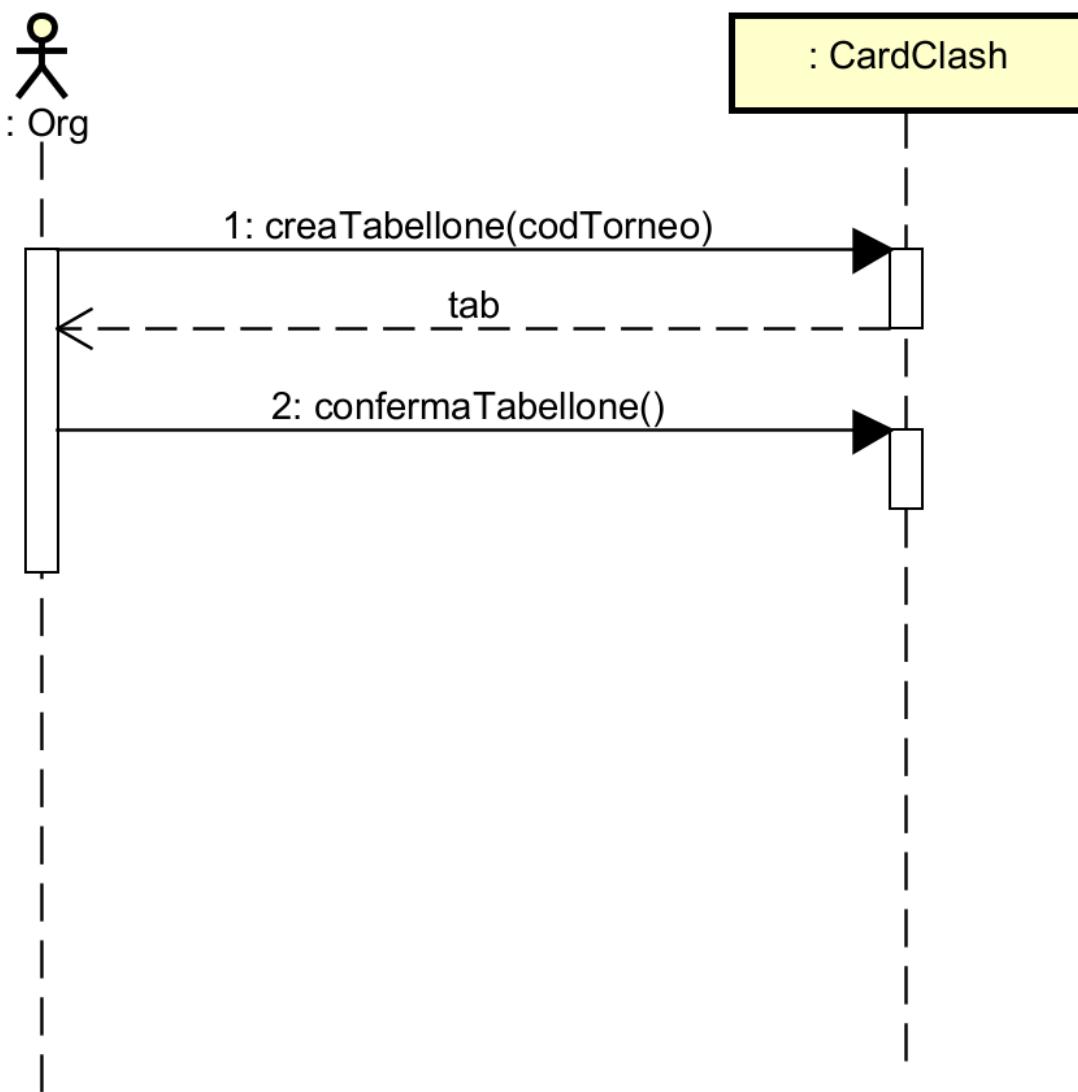
## Contratto3 UC4: Inserimento del tipo del mazzo

- **Operazione:** confermalnserimentoTipo()
- **Riferimenti: Caso d'uso:** Inserimento tipologia Mazzo
- **Pre-condizione:**
  - f è corrente per CardClash
  - tm è corrente per f
- **Post-condizione:**
  - Viene generato e assegnato un codice a tm
  - L'istanza di tm viene salvata e associata correttamente ad f

## 2.2 Diagramma di sequenza di sistema

- **UC5:** Creazione tabellone

Il seguente Diagramma di Sequenza di Sistema (SSD) illustra il corso di eventi I/O tra l'organizzatore e il sistema CardClash.



### 2.2.1 Contratti delle operazioni

I seguenti Contratti descrivono le principali operazioni eseguite dal sistema sulla base degli eventi individuati nell'SSD.

### Contratto1 UC5: Creazione Tabellone

- **Operazione:** creaTabellone(codTorneo)
- **Riferimenti: Caso d'uso:** Creazione Tabellone
- **Pre-condizione:** L'amministratore è autenticato ed ha accesso ai permessi di creazione
- **Post-condizione:**
  - È stata recuperata l'istanza t del torneo selezionato
  - È stata creata un'istanza tab di Tabellone
  - Gli attributi di tab vengono inizializzati
  - tab diventa corrente per torneo
  - I punteggi dei giocatori sono settati a zero.
  - È stato verificato che il numero dei giocatori sia una potenza di 2.
  - Sono state create e istanziate le partite p da disputare
  - Per ogni partita viene generato e settato un codice

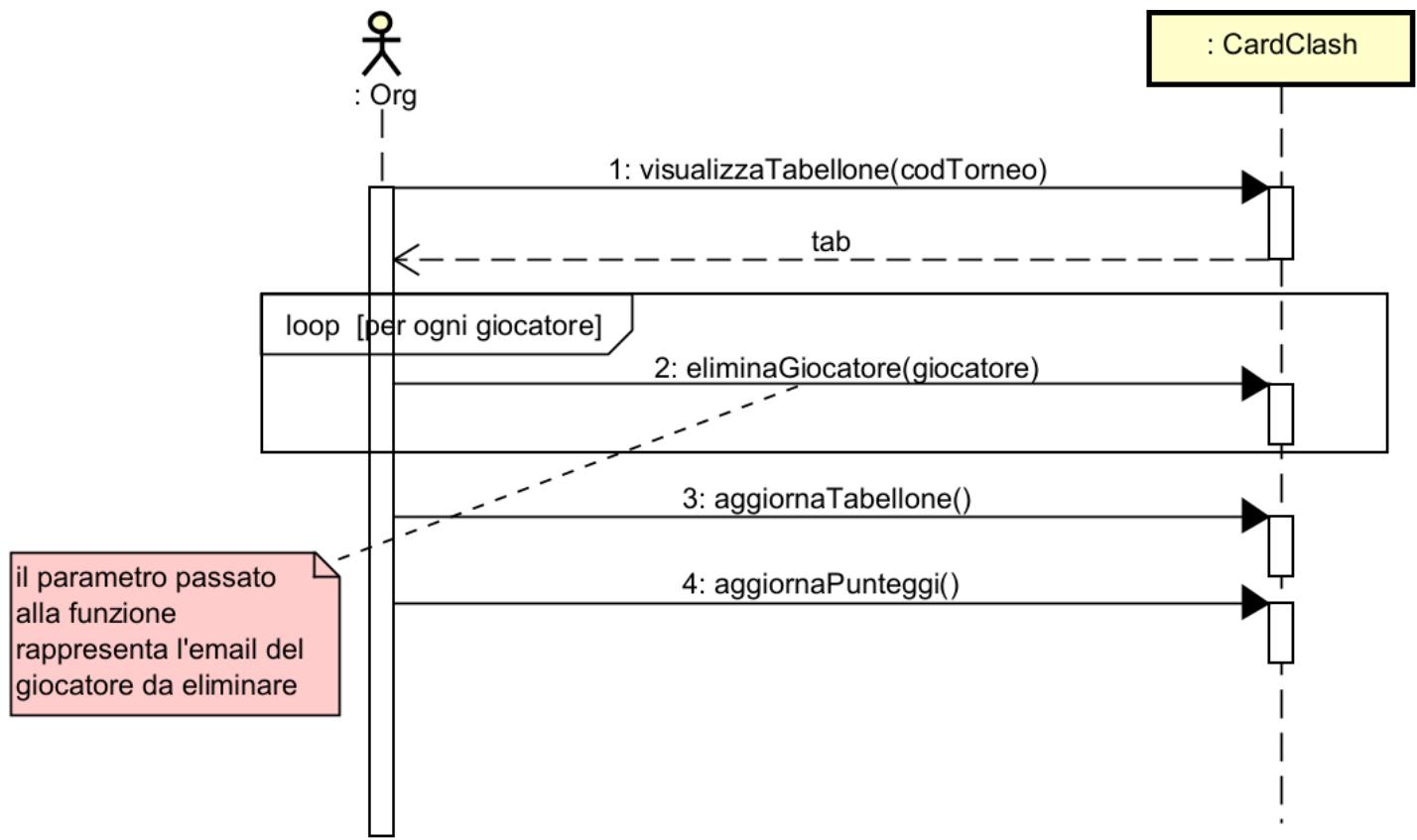
### Contratto2 UC5: Creazione Tabellone

- **Operazione:** confermaTabellone()
- **Riferimenti: Caso d'uso:** Creazione Tabellone
- **Pre-condizione:** gli attributi di tab sono stati inizializzati correttamente e le partite sono state correttamente associate al tabellone.
- **Post-condizione:**
  - Viene generato e associato a tab un id univoco
  - Si salva l'istanza tab di tabellone, e la si associa a t

## 2.3 Diagramma di sequenza di sistema

- **UC6:** Gestisci eliminazione

Il seguente Diagramma di Sequenza di Sistema (SSD) illustra il corso di eventi I/O tra l'organizzatore e il sistema CardClash.



### 2.3.1 Contratti delle operazioni

I seguenti Contratti descrivono le principali operazioni eseguite dal sistema sulla base degli eventi individuati nell'SSD.

#### Contratto1 UC6: Gestione delle eliminazioni

- **Operazione:** `visualizzaTabellone(codTorneo)`
- **Riferimenti: Caso d'uso:** Gestisci eliminazione
- **Pre-condizione:** Sono presenti istanze di tabellone
- **Post-condizione:**
  - Viene recuperata l'istanza `t` del torneo selezionato
  - Viene recuperato il tabellone `tab` associato al torneo `t`

- tab diventa corrente t
- t diventa corrente per CardClash

## Contratto2 UC6: Gestione delle eliminazioni

- **Operazione:** eliminaGiocatore(giocatore)
- **Riferimenti: Caso d'uso:** Gestisci eliminazione
- **Pre-condizione:**
  - Il tabellone è stato recuperato correttamente
  - Il tabellone contiene delle istanze di giocatore
- **Post-condizione:**
  - Il giocatore con e-mail corrispondente a quella passata in input è stato correttamente rimosso dalla lista dei giocatori di tab

## Contratto3 UC6: Gestione delle eliminazioni

- **Operazione:** aggiornaTabellone()
- **Riferimenti: Caso d'uso:** Gestisci eliminazione
- **Pre-condizione:** l'operazione di eliminazione dei giocatori è andata a buon fine
- **Post-condizione:**
  - Vengono create le nuove partite contenti i giocatori ancora in gara e si assegna un codice ad ognuno di esse
  - tab viene settato come nuovo tabellone di t

## Contratto4 UC6: Gestione delle eliminazioni

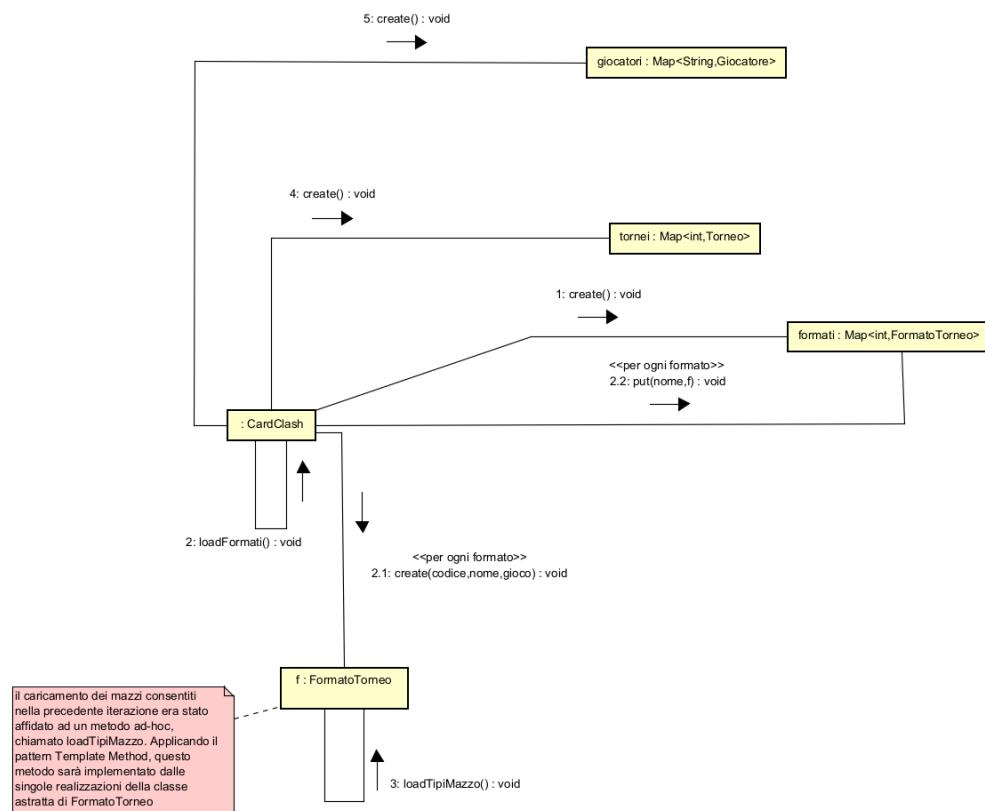
- **Operazione:** aggiornaPunteggi()
- **Riferimenti: Caso d'uso:** Gestisci eliminazione
- **Pre-condizione:** l'operazione di eliminazione è andata a buon fine
- **Post-condizione:**
  - Viene recuperato il formato f di t
  - Viene recuperato il punteggio da assegnare ai giocatori vincitori di questo round del torneo
  - tab aggiorna i punteggi dei giocatori ancora in gara

## 3.0 Fase di Progettazione

Di seguito vengono riportati gli elaborati prodotti. Si vuole porre l'attenzione sull'utilizzo del pattern GoF Template Method all'interno del sistema. Infatti, tale pattern ben si adatta alla situazione che CardClash prevede per la gestione dei Formati dei tornei: tutti i formati di torneo condividono una struttura comune, ma alcune parti della loro implementazione cambiano sulla base della singola istanza; in particolare, questa avrà diverse implementazioni dei metodi: getVictoryScore e getPenaltyScore (secondo le regole di dominio, si veda il documento) e loadTipiMazzo, dato che, come detto più volte, ciascun formato supporta solo certe tipologie di mazzi. Usare ereditarietà con Template Method ci permette di centralizzare la logica comune nella classe astratta e lasciare ai formati specifici solo la personalizzazione delle parti variabili.

### 3.1 Diagramma di comunicazione (caso d'uso di avviamento)

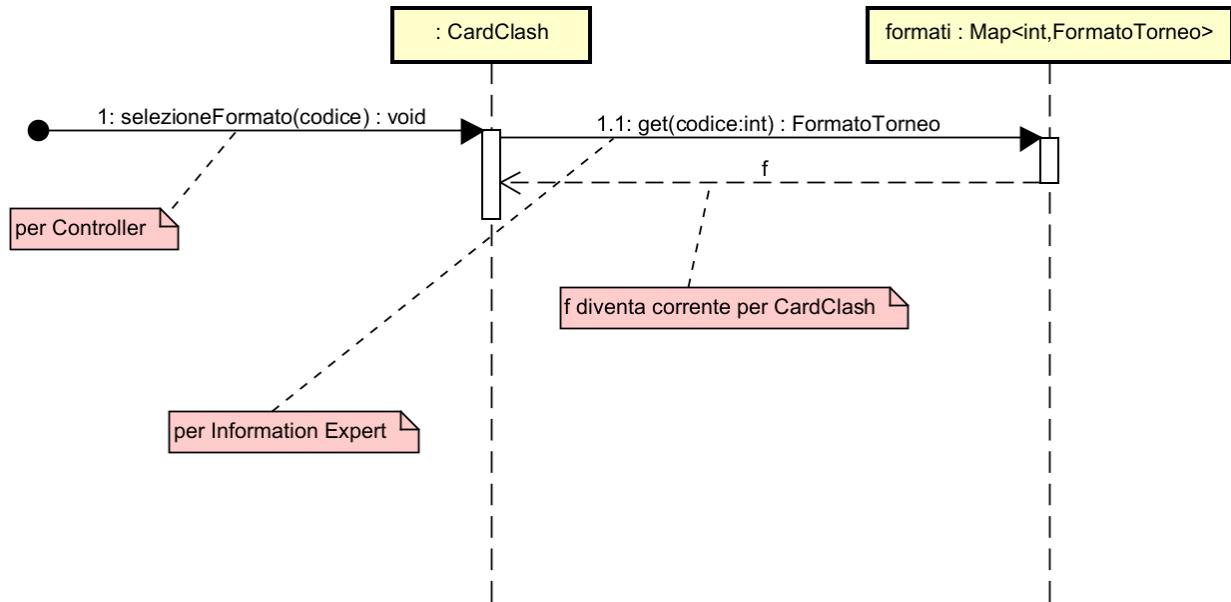
Il caso d'uso di avviamento è stato leggermente modificato rispetto alla prima iterazione per conformarsi meglio alla nuova struttura del dominio del sistema. Si noti il commento circa la diversa implementazione del metodo loadTipiMazzo in accordo con il pattern Template Method:



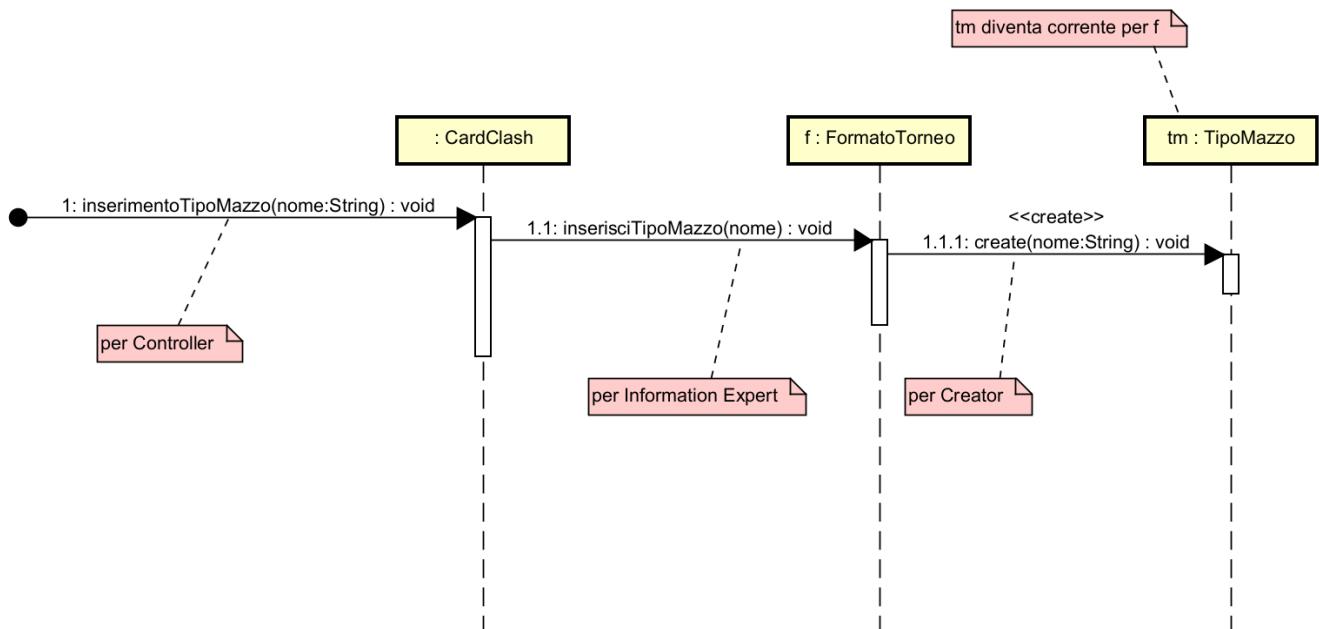
### 3.2 Diagramma di sequenza UC4

Il seguente caso d'uso permette all'amministratore di inserire un nuovo tipo di mazzo per formato di torneo esistente nel sistema.

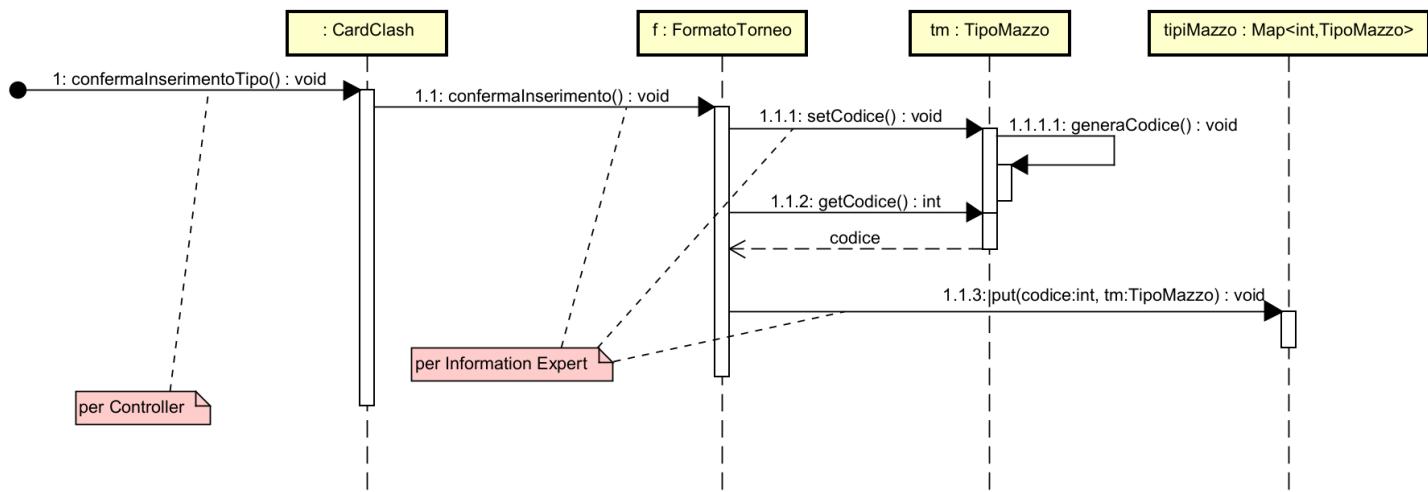
selezioneFormato



inserimentoTipoMazzo



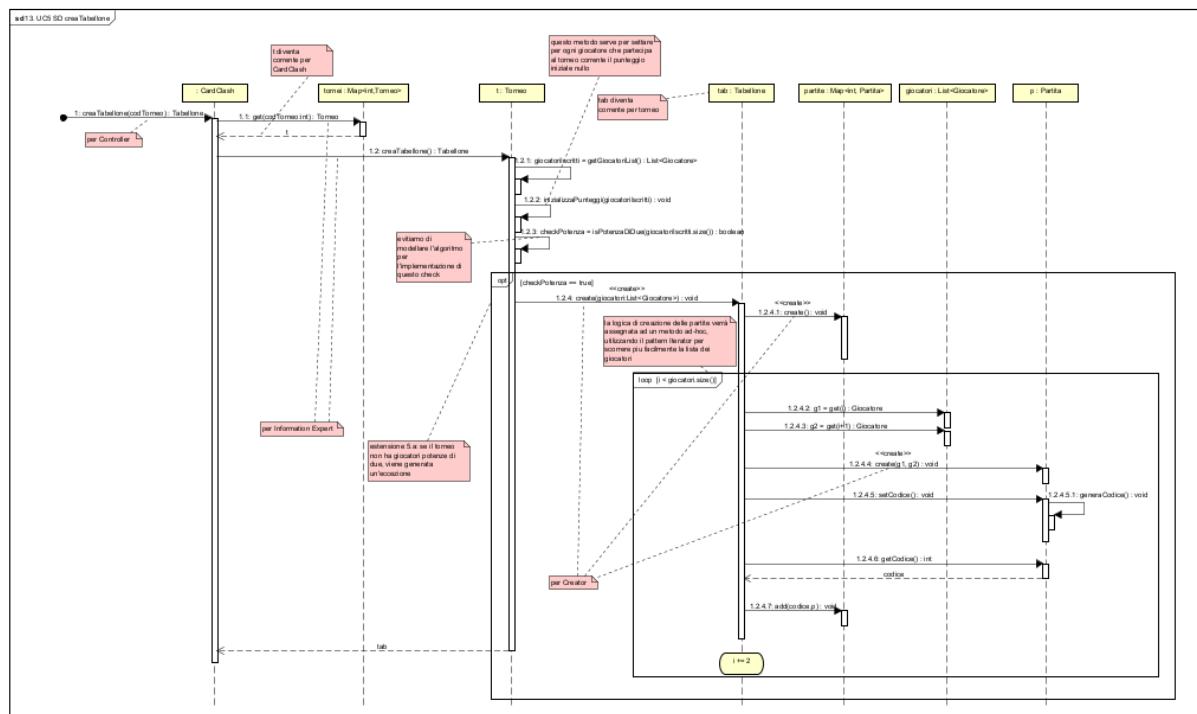
confermalnserimentoTipo



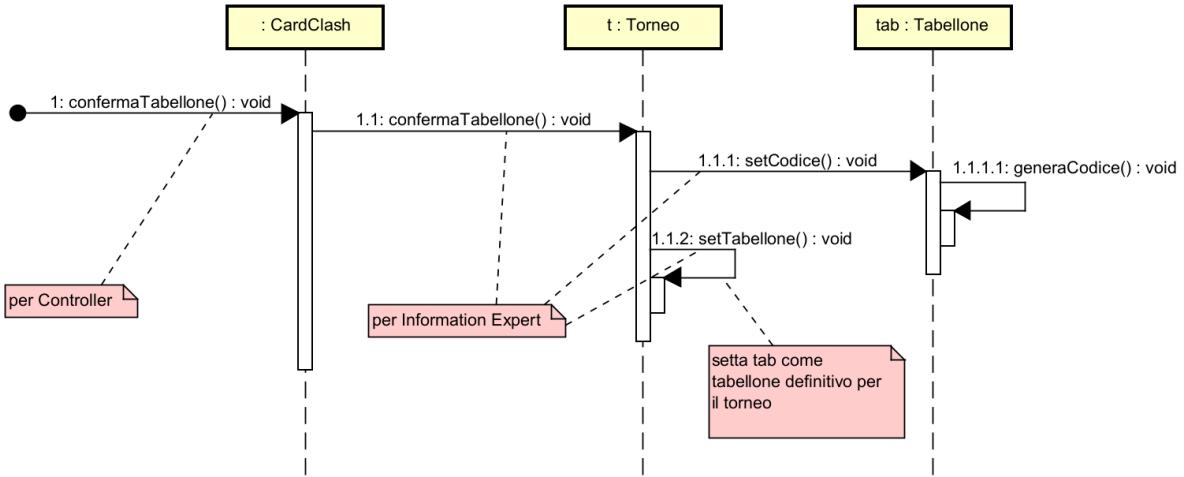
### **3.3 Diagramma di sequenza UC5**

Il seguente caso d'uso permette di generare il tabellone per un torneo selezionato, controllando che il numero di giocatori sia una potenza di 2 per evitare problematiche nella generazione delle partite durante i successivi aggiornamenti del tabellone (estensione 5.a). Vengono anche inizializzati a zero i punteggi di tutti i giocatori in gara. Infine, per la creazione delle partite, la modellazione prevista è stata seguita da un immediato refactoring lato codice, utilizzando infatti un metodo ad-hoc `inizializzaPartite()` possiamo gestire con un'unica funzione anche l'aggiornamento del tabellone, previsto in UC6.

## creaTabellone

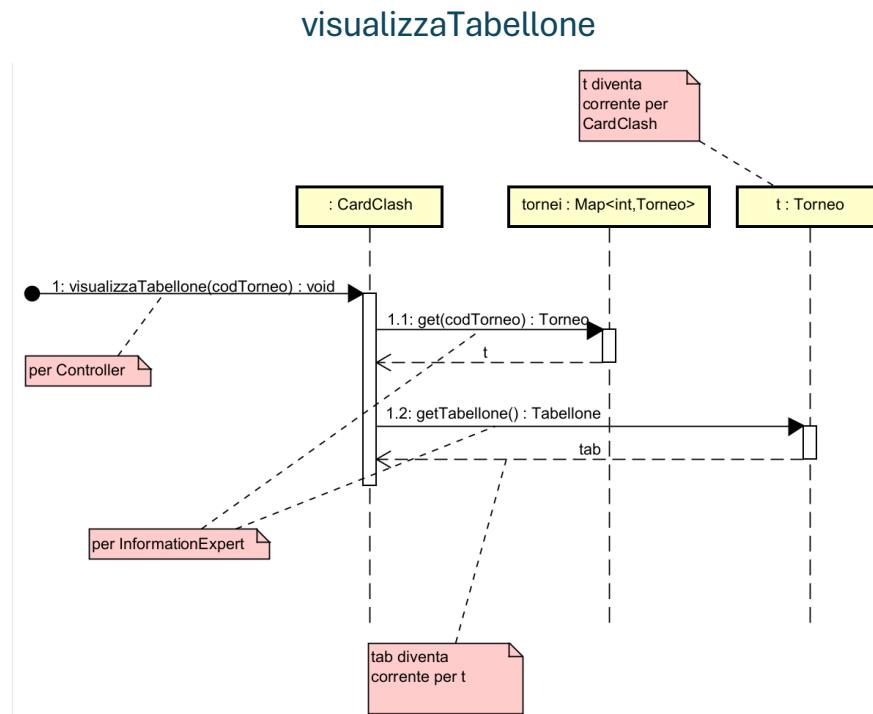


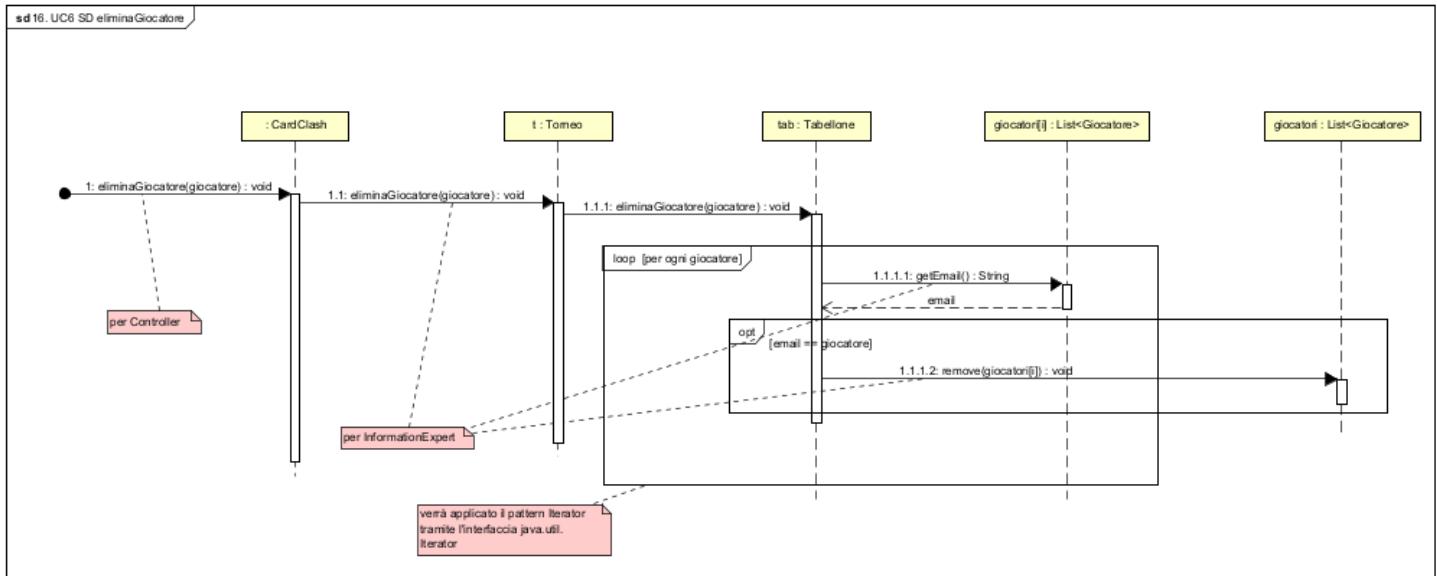
## confermaTabellone



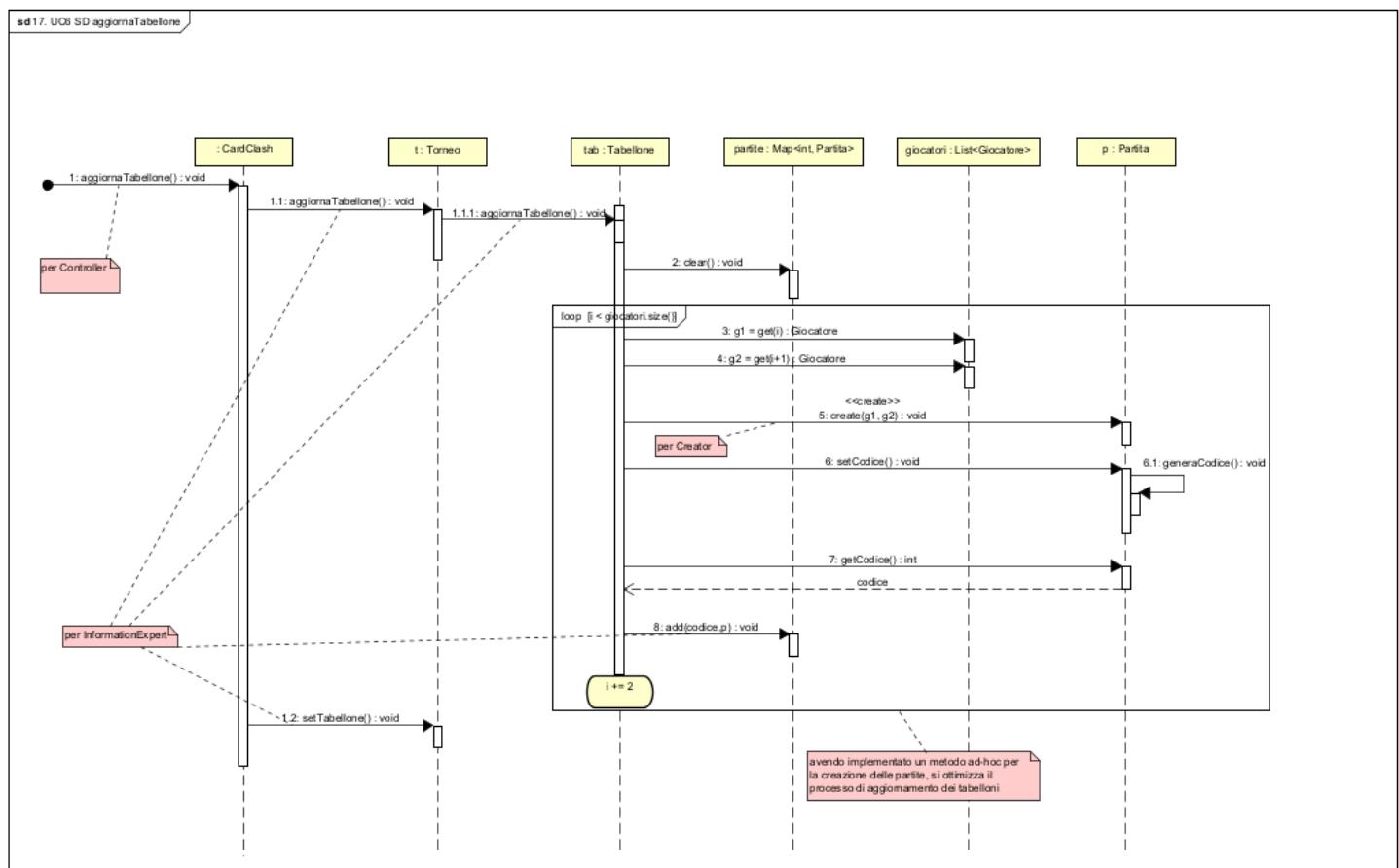
### 3.4 Diagramma di sequenza UC6

Questo caso d'uso è estremamente importante per il corretto funzionamento dell'applicazione, dato che è responsabile dell'esecuzione di momenti cruciali del sistema. Infatti, esso permette non solo di eliminare da un torneo i giocatori che hanno perso le partite della fase precedente, ma permettere di ricreare il nuovo tabellone, con nuove partite, per la fase successiva. Infine, si noti come è stata riutilizzata la stessa logica dell'SD creaTabellone per la creazione delle nuove partite durante l'aggiornamento, che pertanto utilizzerà sempre il metodo inizializzaPartite() creato ad-hoc per questo.

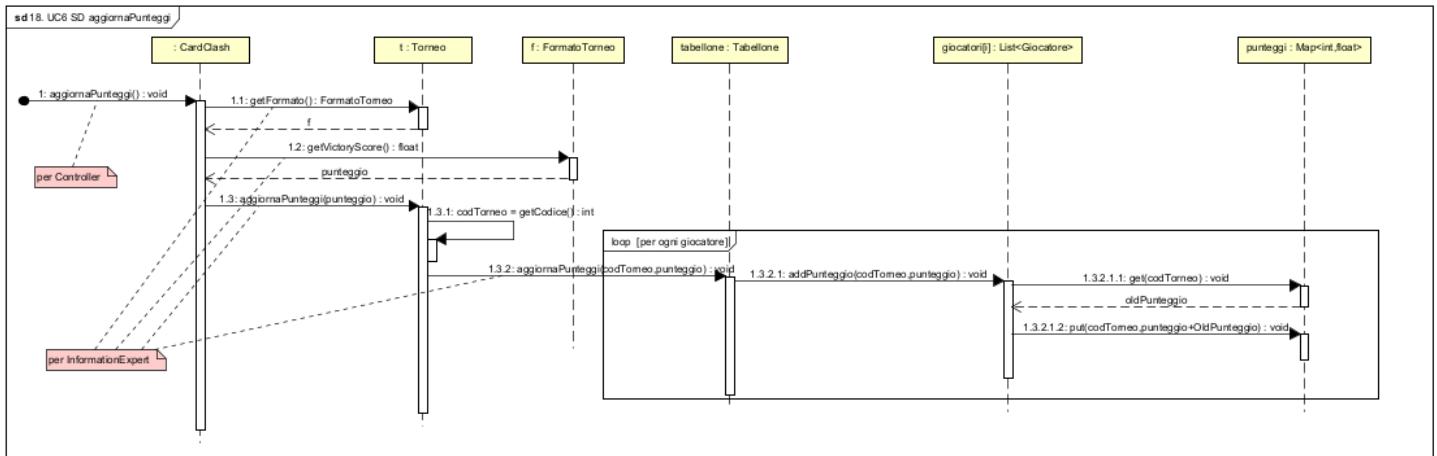




## aggiornaTabellone

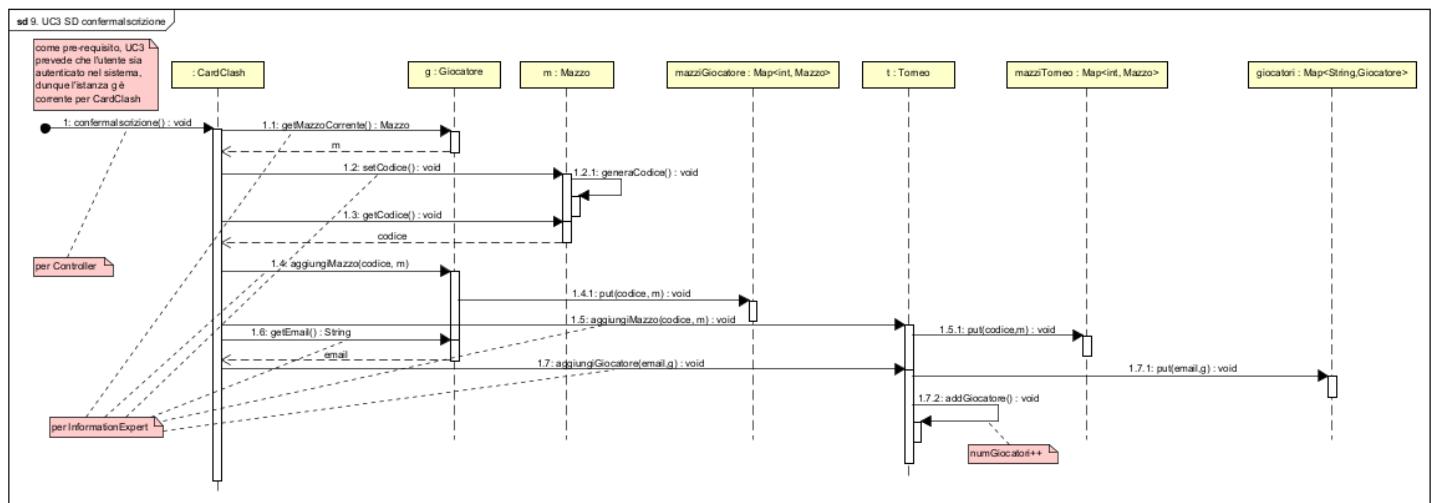


## aggiornaPunteggi



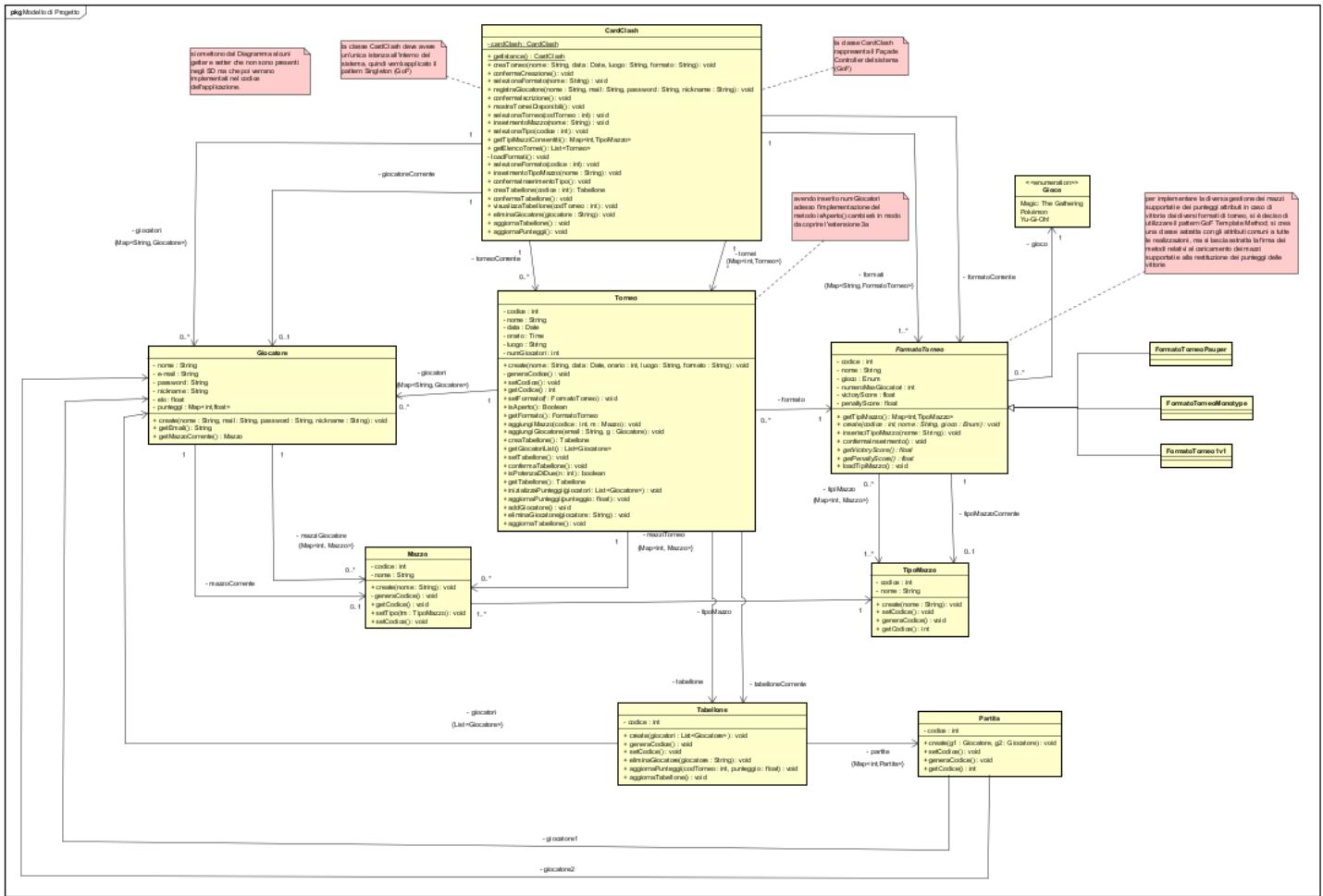
### 3.4 Modifica SD confermaIscrizione (UC3)

Viene incrementato il numero di giocatori iscritti al torneo dopo aver effettuato correttamente l'iscrizione:



## 4.0 Diagramma delle classi

Si riporta il nuovo diagramma delle classi per la corrente iterazione: si pone l'attenzione, come già spiegato in precedenza, sulle modifiche fatto sulla classe FormatoTorneo per applicare il pattern Template Method.



## 5.0 Testing

Durante la seconda iterazione, alcuni metodi di testing definiti nella prima iterazione sono stati rivisti e modificati per adattarsi ai cambiamenti introdotti nel sistema durante questa fase. Inoltre, sono stati testati i nuovi metodi implementati per coprire i casi d'uso della seconda iterazione. Riportiamo di seguito i nuovi metodi di test:

### Card Clash Test

#### 1. testSelezioneFormato()

- Verifica che la selezione di un formato di torneo avvenga correttamente.

#### 2. testInserimentoTipoMazzo()

- Controlla che l'inserimento di un tipo di mazzo nel formato selezionato sia corretto. Dopo aver selezionato un formato e aggiunto un tipo di mazzo, il test verifica che il formato contenga effettivamente il tipo di mazzo appena inserito e che il nome sia stato assegnato correttamente.

### **3. `testConfermaInserimentoTipo()`**

- Assicura che, dopo l'inserimento di un nuovo tipo di mazzo, la conferma dell'inserimento aggiorni correttamente la lista dei tipi di mazzo disponibili. Il test verifica che la dimensione della lista sia aumentata di 1 e che il nuovo tipo sia presente all'interno del formato.

### **4. `testCreaTabellone()`**

- Verifica che la creazione di un tabellone per un torneo funzioni correttamente.

### **5. `testConfermaTabellone()`**

- Verifica che la conferma del tabellone di un torneo sia stata eseguita correttamente, assicurandosi che il tabellone confermato appartenga effettivamente al torneo corrente.

### **6. `testVisualizzaTabellone()`**

- Verifica che il tabellone del torneo corrente possa essere visualizzato correttamente.

### **7. `testEliminaGiocatore()`**

- Verifica che, quando un giocatore viene eliminato, il tabellone del torneo corrente non contenga più la relativa istanza.

### **8. `testAggiornaTabellone()`**

- Verifica che l'aggiornamento del tabellone avvenga in maniera corretta, verificando che il tabellone una volta aggiornato sia ancora presente nel torneo corrente.

## **9. testAggiornaPunteggi()**

- Verifica che, alla creazione di un tabellone, il giocatore abbia un punteggio iniziale di default pari a 0.0f. Successivamente, controlla che l'aggiornamento del punteggio avvenga correttamente in base alle regole del formato del torneo.

### **GicatoreTest**

#### **1. testSetPunteggio()**

- Verifica che il giocatore abbia il punteggio di default pari a 0.0f quando viene invocato il metodo di setPunteggio.

#### **2. testAddPunteggio()**

- Verifica che al giocatore sia attribuito correttamente il punteggio previsto.

### **TabelloneTest**

#### **1. testCreazioneTabellone()**

- Verifica che il tabellone venga creato correttamente, e che il numero di giocatori presenti per il tabellone creato sia pari a 4.

#### **2. testInizializzaPartite()**

- Verifica che, con 4 giocatori, il numero di partite inizializzate sia esattamente 2. Inoltre, controlla che ogni partita includa correttamente i giocatori assegnati.

#### **3. testEliminaGiocatore()**

- Verifica che, eliminando 2 giocatori da un torneo con 4 iscritti, la lista dei giocatori rimanenti sia esattamente composta da 2 elementi.

#### **4. testAggiornaTabellone()**

- Verifica che, eliminando 2 giocatori da un torneo con 4 iscritti, una volta invocato il metodo di aggiornamento del tabellone il numero delle partite rimanenti sia esattamente pari a 1.

## **5. testAggiornaPunteggi()**

- Verifica che, una volta che i punteggi sono stati aggiornati per il tabellone del torneo corrente, ogni giocatore all'interno dell'ArrayList dei giocatori rimanenti abbia il punteggio aggiornato.

## **TorneoTest**

### **1. testCreaTabellone()**

- Verifica che la creazione del tabellone per il torneo avvenga correttamente. Dopo aver invocato il metodo creaTabellone(), il test controlla che il tabellone non sia nullo e che il numero di partite generate sia pari a 2, confermando che la struttura del tabellone è stata creata correttamente.

### **2. testCreaTabelloneGiocatoriNonPotenzaDiDue()**

- Assicura che, se il numero di giocatori iscritti al torneo non è una potenza di due, venga sollevata un'eccezione. Il test aggiunge un giocatore extra al torneo e verifica che il tentativo di creare un tabellone in questa condizione lanci effettivamente l'eccezione attesa.

### **3. testConfermaTabellone()**

- Controlla che la conferma del tabellone avvenga correttamente. Dopo aver creato il tabellone con creaTabellone(), viene invocato confermaTabellone(). Il test verifica che il tabellone confermato sia effettivamente associato al torneo, assicurando che l'operazione sia andata a buon fine.