

# PROTOCOLLO DI RETE

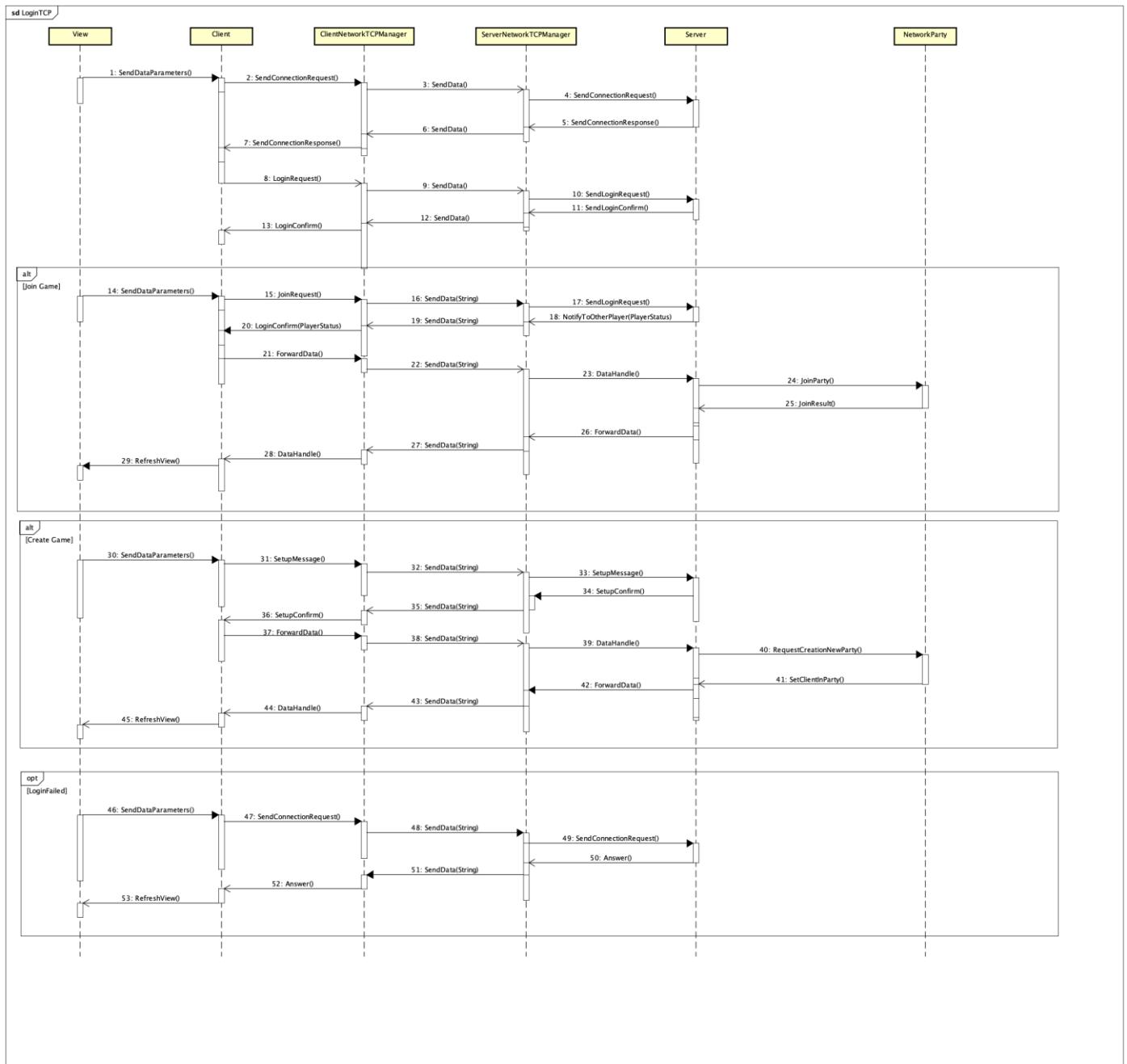
## Gruppo AM40

### Introduzione

Per descrivere il protocollo di rete è necessario prima suddividere gli stati di gioco in quattro macro fasi considerate "critiche" dal punto di vista dell'interazione tra Client e Server. Le fasi sono le seguenti: **Login**, **FirstRound**, **Round** e **EndGame**.

Nota: Nell'implementazione TCP viene utilizzato JSON per la serializzazione degli oggetti. Quindi, alcuni dati JSON dentro al Sequence diagram verranno sostituiti da "...", cioè un payload di dati che, altrimenti, renderebbe troppo pesante il diagramma alla vista. Qualora i dati non fossero intuitivi a priori, sotto ad ogni macro fase descritta in questo file viene proposta una guida per identificare il Payload di dati.

### COMUNICAZIONE TCP



## **LOGIN**

### **Descrizione**

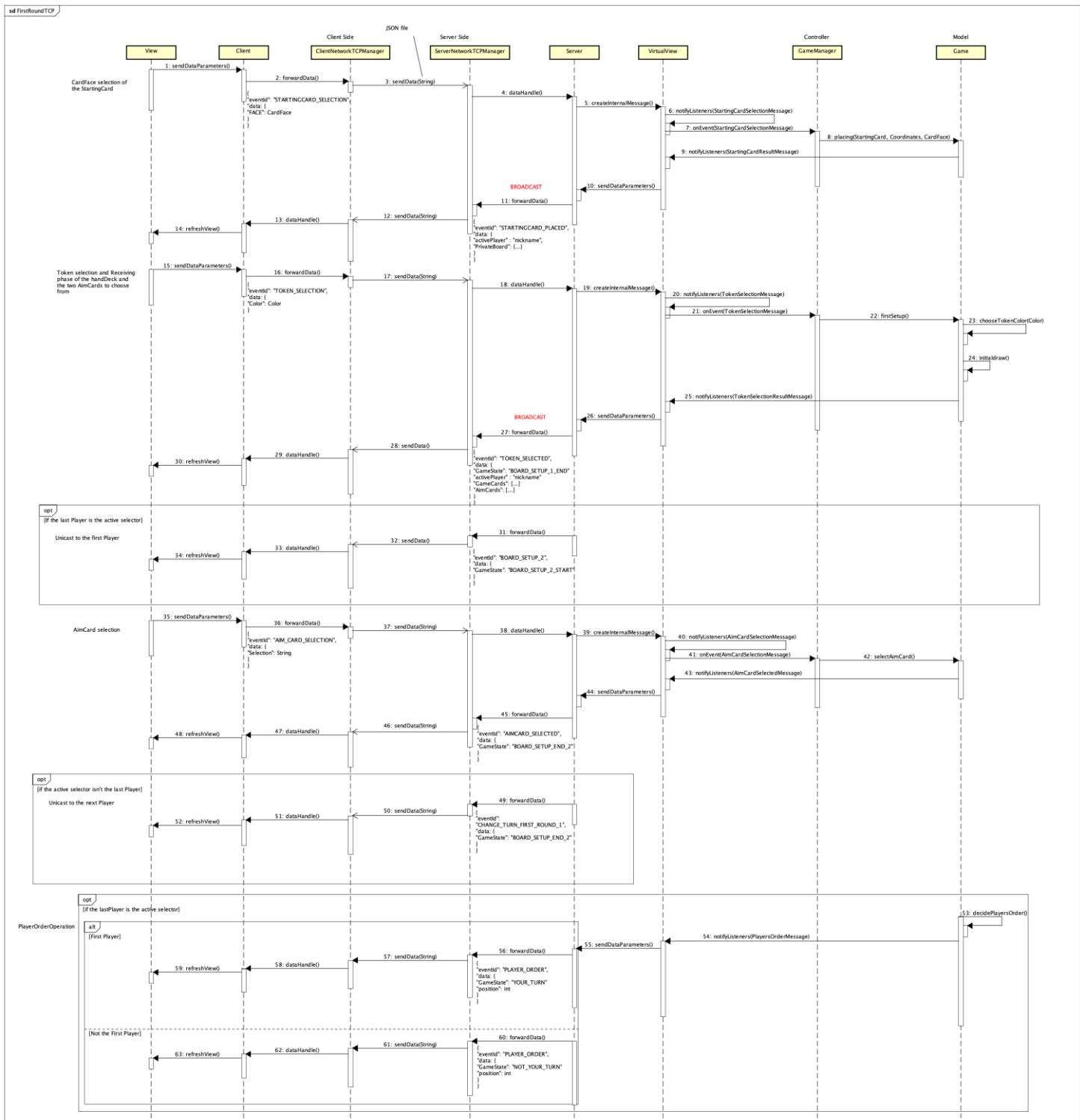
Login rappresenta il flusso di comunicazione iniziale per la connessione tra Client e Server attraverso l'utilizzo del protocollo TCP, che prevede l'uso di ClientNetworkTCPManager e di ServerNetworkTCPManager come classi mediatiche per la comunicazione. La fase di login si tramuta in un set up a livello di rete, che colloca i giocatori in uno stato di "lobby astratta", nella quale i giocatori, in base alla scelta di unirsi ad una partita già creata o creare una nuova partita, verranno assegnati ad un NetworkParty; una volta raggiunto il numero di giocatori scelto dal Player iniziale si passerà alla prossima fase.

Il Client, nella fattispecie, invierà prima di tutto una richiesta di connessione al server, che, in base al caso in questione, darà un riscontro variabile:

- 1) Nel caso di una richiesta di connessione non conforme, si entra in un loop fino alla prima richiesta corretta;
- 2) Nel caso di una richiesta corretta il Server avvisa il Client con la conferma di connessione avvenuta. A questo punto il client invierà una seconda richiesta al Server per effettuare il login e quest'ultimo invierà messaggio di conferma. A login avvenuto, ci saranno due casi:
  - 1) Ci sono dei NetworkParty già esistenti in attesa del raggiungimento del numero di Players stabilito per l'inizio della partita, quindi il Client potrà inviare la richiesta di Join al server e una volta entrato in partita avrà un suo identificativo; gli altri Players già presenti saranno notificati dell'accesso di un nuovo player; a questo punto se il nuovo Player è l'ultimo necessario a far avviare la partita, i Players verranno avvertiti che la partita può iniziare.
  - 2) In alternativa, per creare una nuova partita, il Client invierà un messaggio di setup al Server e dopo aver trasmesso i dati, dal server partirà un messaggio verso il NetworkParty con la chiamata a metodo per la creazione di un nuovo Party, con conseguente aggiunta del Client che ha inviato la richiesta.

A questo punto la partita può iniziare e si inizializza il Model attraverso Game per passare alla fase di FirstRound.

Per mantenere pulizia visiva e coerenza con la fase, viene omesso il messaggio in cui si inizializza il Game e lo stato del gioco viene impostato a BOARD\_SETUP\_1



## FIRST ROUND

### Descrizione

Da questo punto in poi verrà sfruttato a pieno il meccanismo di poster e listener presente nel MVC, con interazione “triangolare” tra Virtual View, Controller e Model a seguito della ricezione di un input dal Client.

Lato Client, una volta ricevuti i parametri dalla View, con `forwardData()` viene creato il file JSON che racchiuderà dentro di sé le informazioni necessarie per informare il Controller degli input da passare come parametro ai metodi di interfaccia del Model. Una volta che il Json arriva al Server viene fatto parsing del messaggio e viene creato il corrispettivo messaggio interno all'MVC per il meccanismo ad eventi MVC. Questo è il “flusso” generale da qui in avanti.

Entrando nel dettaglio di questa fase, FirstRound rappresenta il set up iniziale del gioco, ovvero la fase in cui ogni giocatore prepara e sceglie i componenti di gioco associati al Player; ciò implica la decisione dell'orientamento delle Starter Cards, la scelta del colore del segnalino e l'acquisizione delle 2 Resource Cards e della Gold Card. Questa è la prima sotto fase del FirstRound. Infine, una volta superata la prima sotto fase da ogni client, viene notificato il primo client in ordine di selezione dell'evoluzione dello stato. Quindi si prosegue e ogni giocatore sceglie la Aim Card che preferisce tra le due ricevute nel messaggio di risposta del server di aggiornamento della propria situazione nella fase precedente, assieme alla risposta del colore del Token selezionato.

N.B.

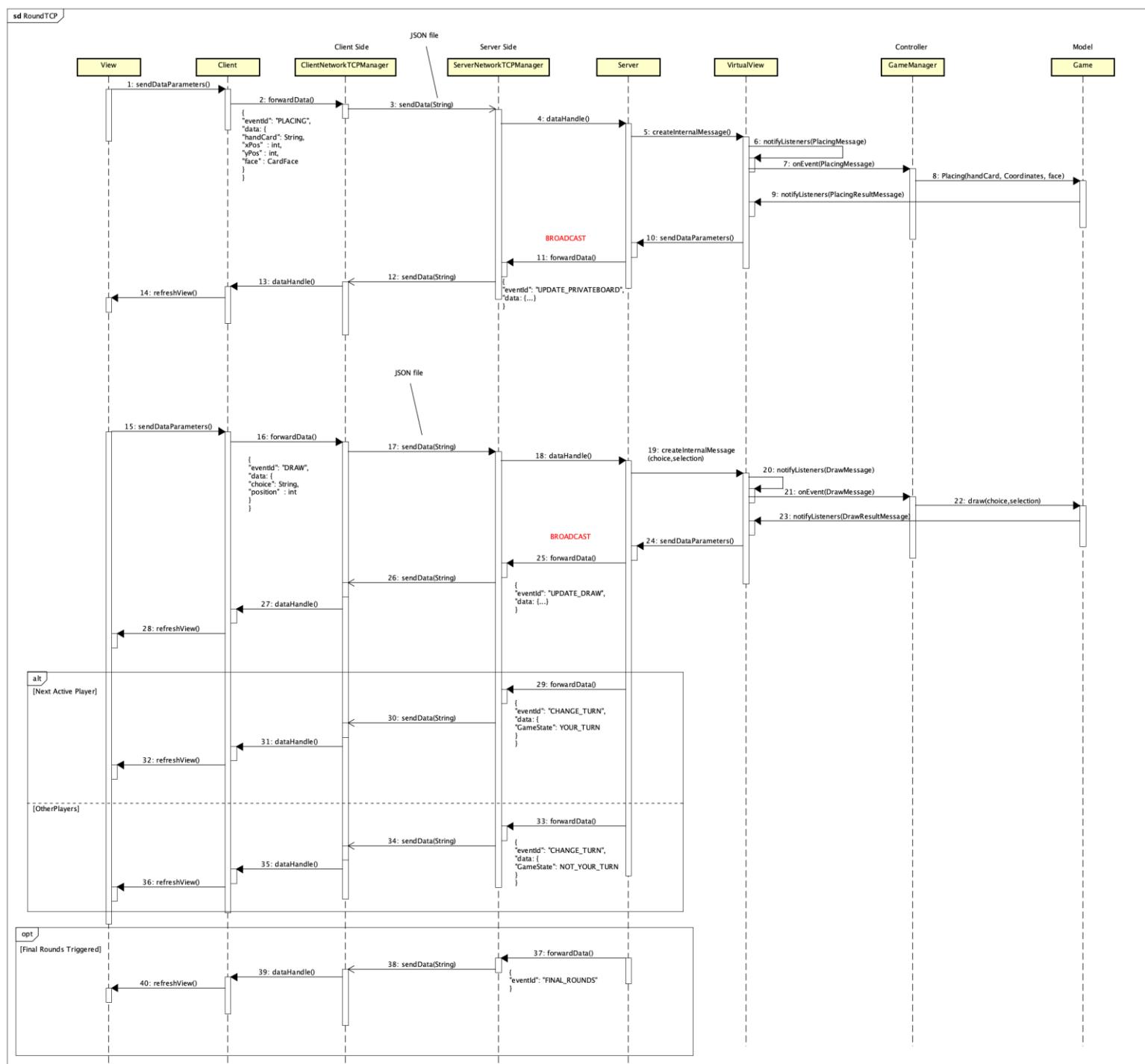
Generalmente i messaggi broadcast sono quelli inviati dal Server come risposta con il metodo `forwardData()` in seguito all'inserimento corretto dei dati in input da parte dell'utente. Il caso in cui l'input non viene considerato valido non è rappresentato nel sequence diagram per alleggerire la visualizzazione. Comunque, in tal caso, viene lanciata un'eccezione dal modello che si tramuta in un messaggio di risposta che identifica l'inserimento non valido. Quindi il client dovrà reinserire l'input finché non sarà considerato valido per far evolvere lo stato del modello. Nel caso si voglia vedere un esempio per la gestione di un input non valido, si guardi nella sezione ROUND in `25.forwardData() di draw`

Infine, una volta che il FirstRound giunge al termine, viene stabilito casualmente l'ordine dei giocatori come da regolamento e i giocatori vengono notificati con il proprio ordine di gioco subito dopo la risposta del server alla ricezione dell'input dell'ultimo Player che seleziona l'Aim Card personale.

#### - Payload dati JSON:

**11. `forwardData () —> Privateboard {..}`** racchiude dentro di sé il counter degli elementi presenti sulla board di un giocatore, le coordinate possibili per i futuri piazzamenti, l'ID della carta piazzata e l'orientamento della carta

**27. `forwardData () —>`** sebbene sia abbastanza autoesplicativo come messaggio, si sottolinea il fatto che ogni qualvolta si debba inviare una carta per messaggio, viene passato solo l'ID della Carta visto che, una volta che il messaggio arriva al client, viene parzialmente ricostruita la carta a partire dall'ID prendendo i parametri collegati e utili alla visualizzazione direttamente dal file JSON dove sono raccolte tutte le carte di gioco



## ROUND

### Descrizione

Round rappresenta la fase di round che si ripete ciclicamente per ogni giocatore finché non avviene il trigger per l'endgame.

Un giocatore inizia piazzando una carta dalla sua mano fornendo coordinate, presumibilmente tra quelle presenti ricevute dal server nella fase precedente, poi sceglie una faccia su quale piazzare la carta. In seguito il server risponde con esito positivo o negativo. Anche qui, come per FirstRound, per semplicità visiva è stato omesso il caso in cui un input non sia valido, però il funzionamento rimane il medesimo a quello menzionato prima.

A seguito di un piazzamento considerato valido, vengono notificati tutti i giocatori delle modifiche sulla PrivateBoard del giocatore e si prosegue con la fase di pescaggio (draw).

Nella fase di draw è necessario che l'utente fornisca due parametri per capire la differenza tra il tipo di carta che vuole pescare (resource oppure golden) e da dove pescarla (deck oppure plate). Come nei casi precedenti, sul diagramma viene omesso il caso in cui venga lanciata un'eccezione e sia necessario reinserire l'input.

#### - Payload dati JSON

##### PLACING

**11. forwardData ()** —> Allo stesso modo di FirstRound, data {...} racchiude dentro di sé la modifica al counter degli elementi aggiunti con la nuova carta posizionata, le nuove coordinate per possibili piazzamenti futuri, l'ID della carta piazzata, l'orientamento della carta e i punti acquisiti da un giocatore dopo il posizionamento della carta (se una carta, per qualsiasi motivo, non dovesse dare punti, il dato ha valore 0)

##### DRAW

**16. forwardData ()** —> Il parametro "choice" sarà colui che definisce quale tipo di carta l'utente vorrà pescare, quindi potrà assumere i valori "resource" oppure "golden". Il parametro "position" sarà quello che definirà da dove l'utente vorrà pescare la carta, quindi potrà essere "deck", "first" (prima carta del plate), "second" (seconda carta del plate) e verrà fatta la conversione ad int della scelta fatta dall'utente per che farà assumere i valori first=0, second=1, deck=2 (è più comodo per il modello) al fine di ottenere subito la posizione nell'array che rappresenta il plate della carta selezionata.

**25. forwardData ()** —> per rendere l'idea di gestione dell'eccezione, il messaggio di ritorno sarà un update dello stato del gioco (se il processo è andato a buon fine) oppure un messaggio di ripetizione della pescata.

Il messaggio UPDATE\_DRAW, quindi, nel sequence in realtà rappresenta:

- 1- Caso negativo, la pescata non è andata a buon fine

```
{  
    "eventID": "REPEATDRAW",  
    "data": {  
        "errorDraw": String  
    }  
}
```

- 2- Caso positivo, la pescata è andata a buon fine

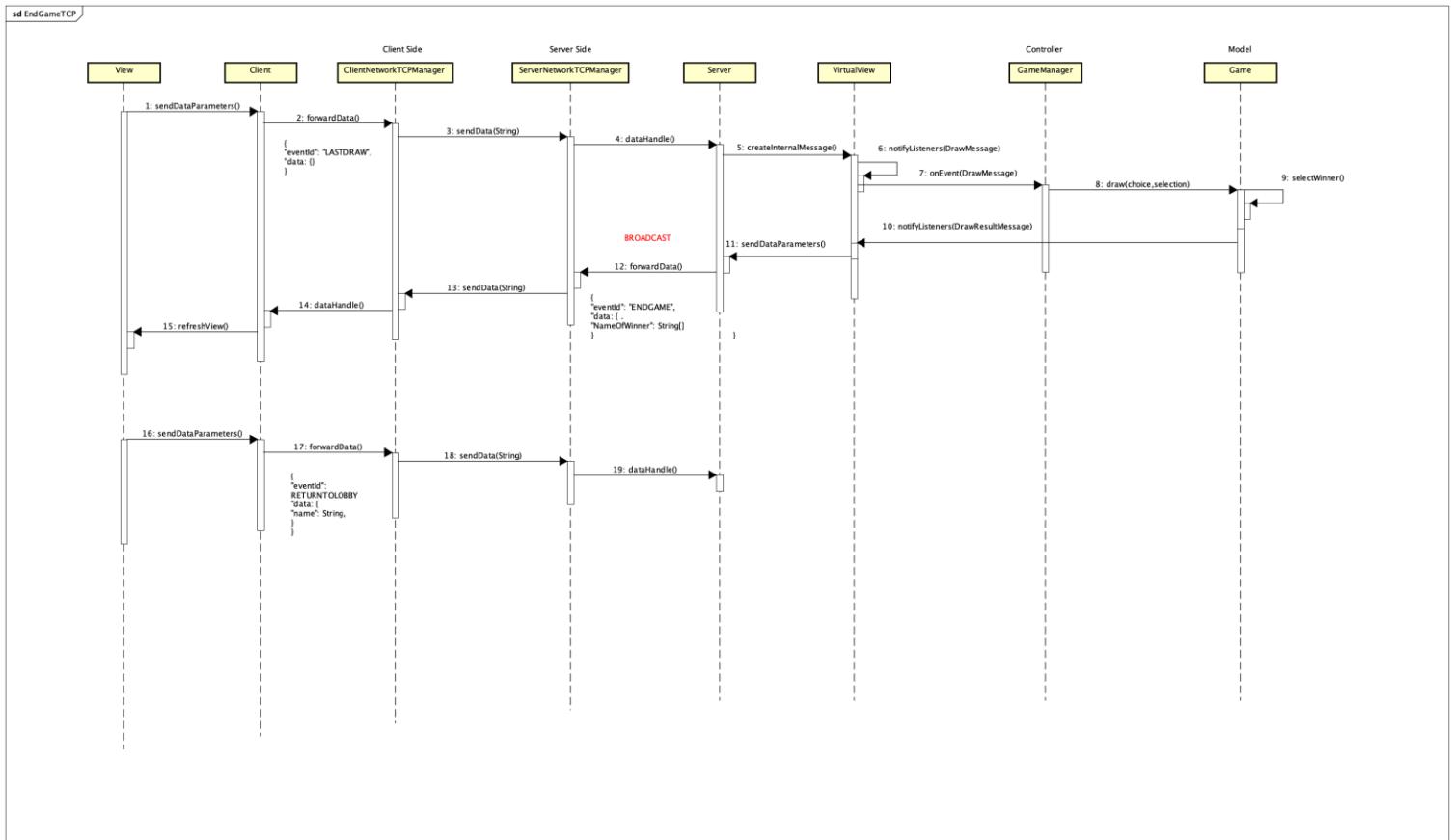
```
{  
    "eventID": "POSITIVEDRAW",  
    "data": {  
        "senderNickname": String,  
        "CardPlateID": int,  
        "cardDrewID": int,  
    }  
}
```

}

In base a ciò che avviene nel Model, si genera un messaggio che rappresenta errore oppure che l'operazione è andata a buon fine. Tale messaggio è una sottoclasse della classe DrawResultMessage a cui viene associato via creazione del file JSON da spedire al Client il corrispettivo eventID con il relativo payload ricavando i dati dal tipo dinamico del messaggio interno.

In caso positivo, il parametro "CardPlateID" rappresenta l'ID della nuova carta che c'è sul plate se l'utente ha pescato dal plate.

Il parametro "CardDrawID" rappresenta l'ID della carta pescata dall'utente.



## ENDGAME

### Descrizione

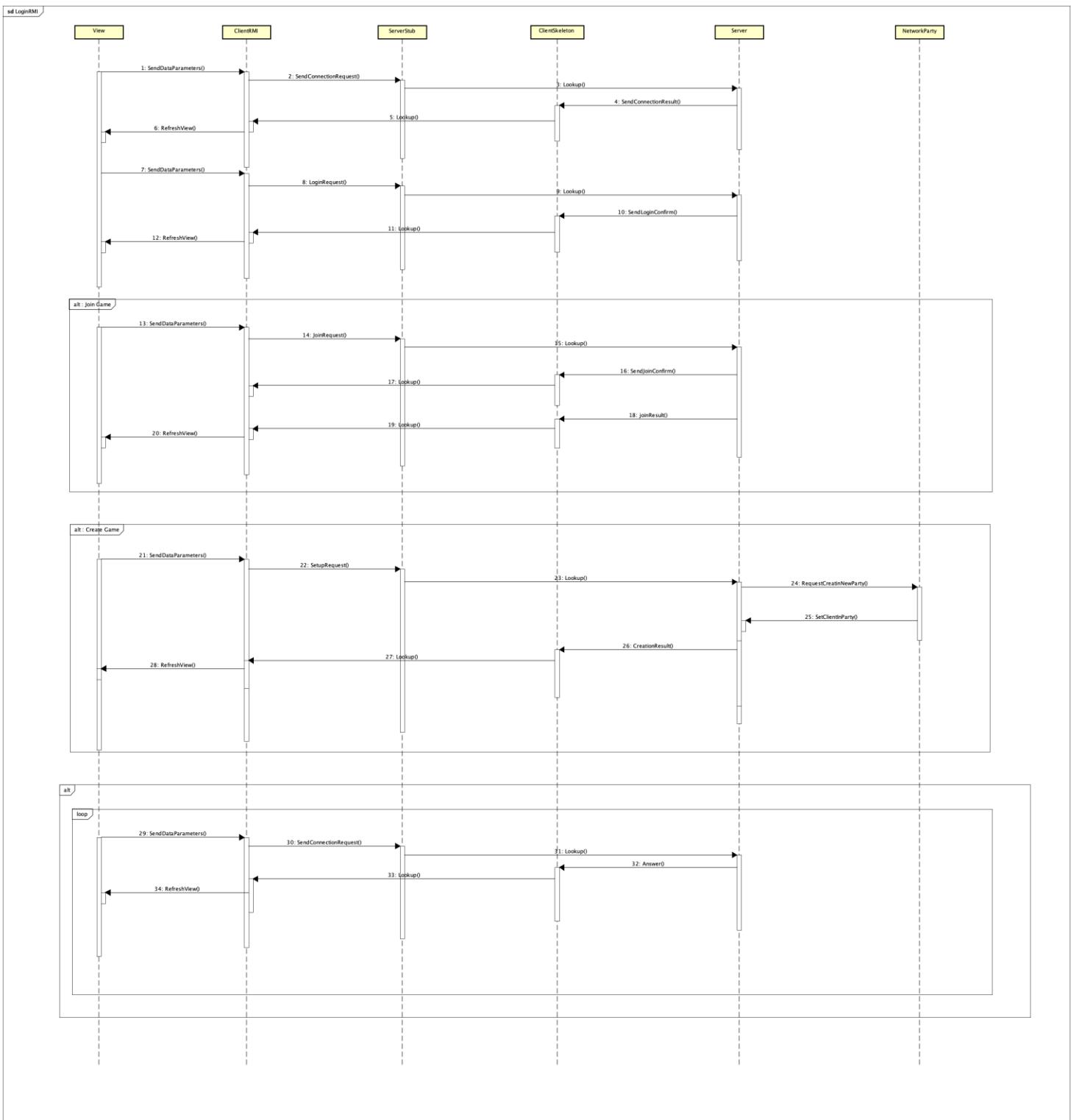
Questa fase si ricollega a quando l'ultimo giocatore nell'ordine di gioco pesca la sua carta e al contempo nel ciclo di round precedente è avvenuto un trigger della fase di Endgame, a seguito della raggiunta di una delle condizioni descritte dal regolamento.

Quindi, una volta che i giocatori hanno giocato il loro ultimo turno e l'ultimo giocatore ha concluso il suo round, il server, in risposta all'ultima draw manda, in aggiunta alla classica risposta del server ad una draw, anche un messaggio in broadcast con, al suo interno, il vincitore o i vincitori della partita. Alla fine della partita il giocatore viene riportato in lobby.

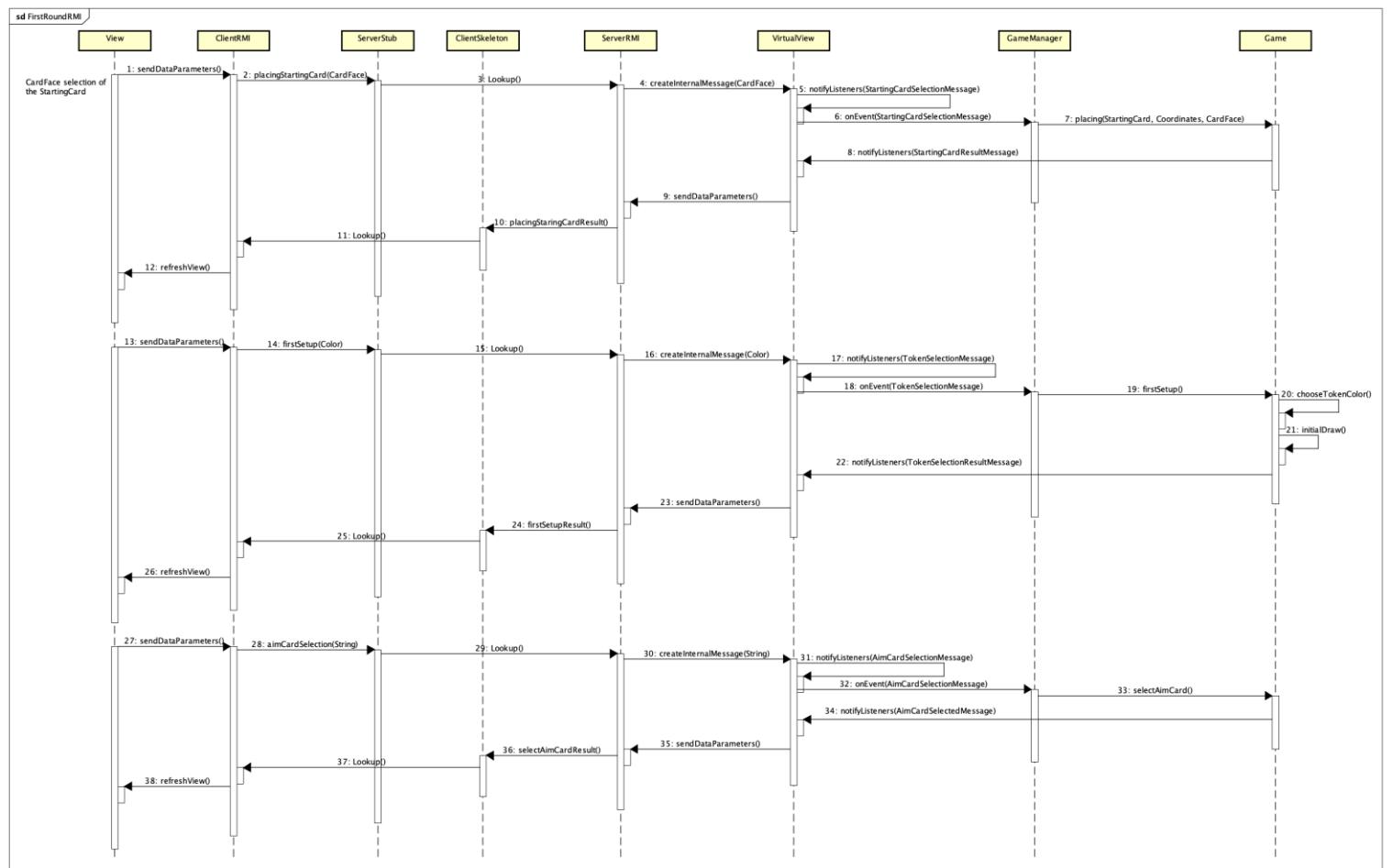
## **COMUNICAZIONE RMI**

Il meccanismo di funzionamento di RMI è parallelo a quello di TCP, la differenza sostanziale rimane nel fatto che nel Client non viene più creato un messaggio da mandare via rete, ma vengono passati direttamente i parametri di input al metodo invocato da remoto attraverso ServerStub che si collega direttamente alla Virtual View dal Server e, quindi, al Controller e al Model.

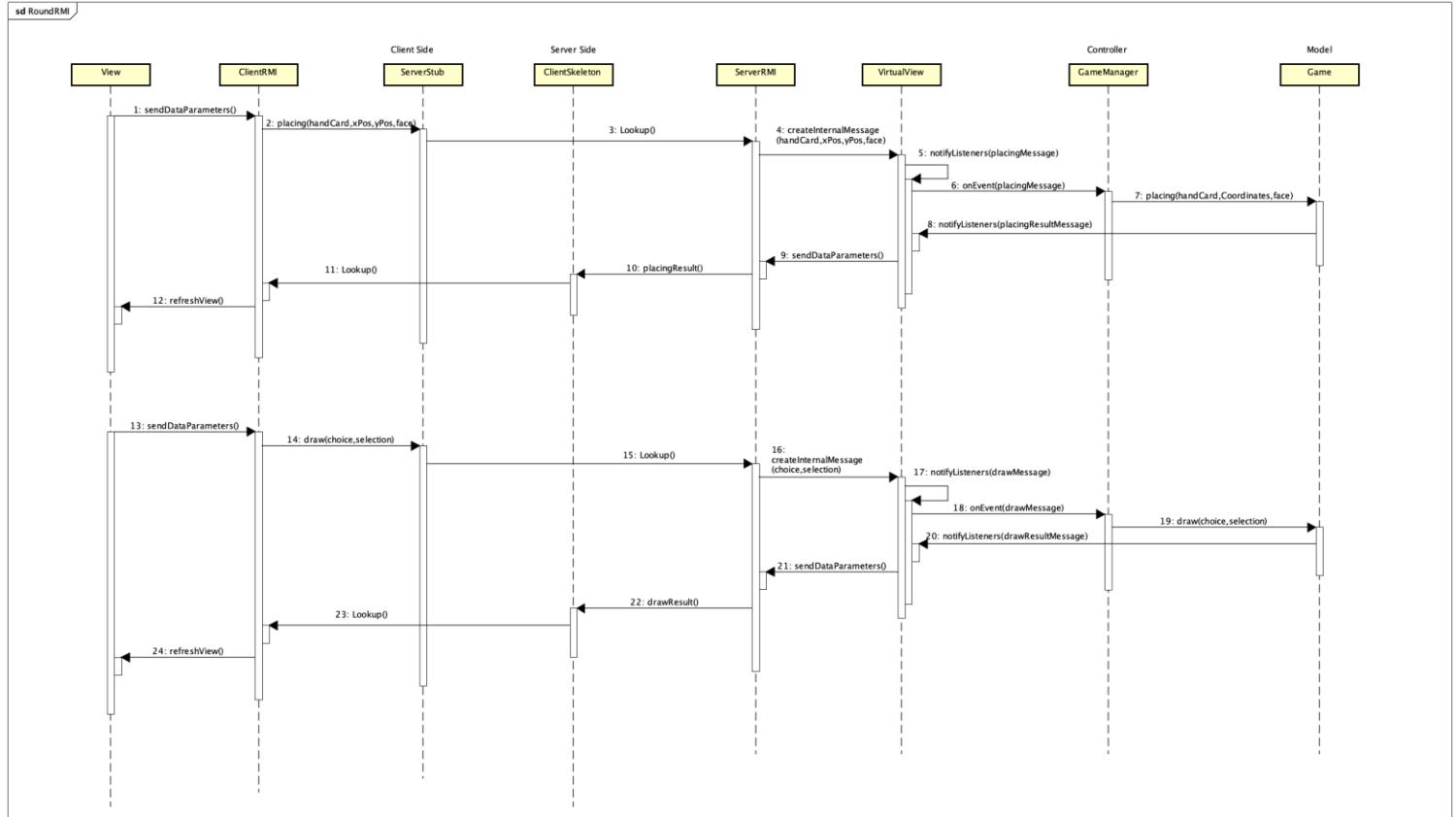
La risposta del server avviene in maniera analoga utilizzando, però, ClientSkeleton e, anche in questo caso, viene sottintesa la gestione dell'errore in caso di input non valido che viene generato tramite eccezione dal Model e viene tradotta in un comportamento variabile all'interno del metodo chiamato.



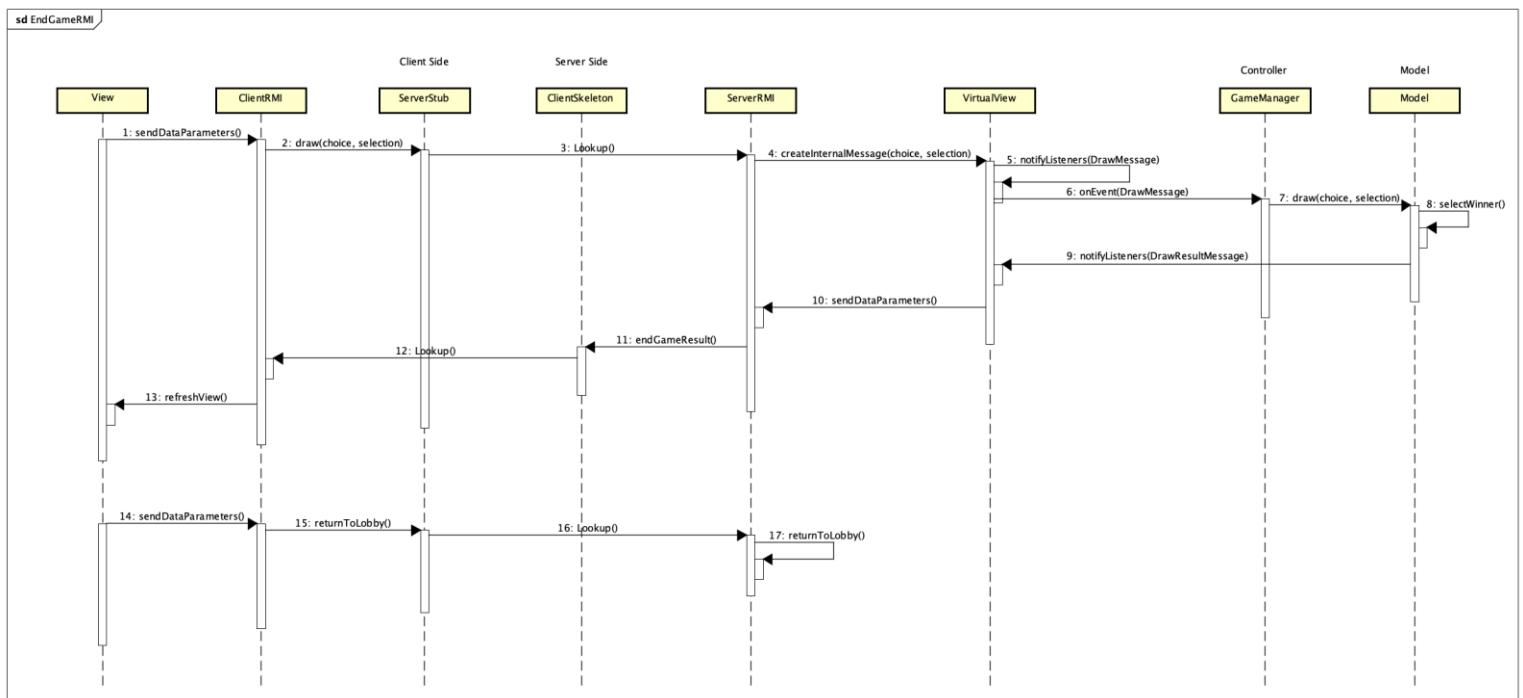
## LOGIN



## FIRST ROUND



## ROUND



**ENDGAME**