Tokyo - progetto di Sistemi Distribuiti

Beatrice Bacelli Luca Bedogni Silvia Righini

18 giugno 2010

1 Il gioco: Tokyo

La nostra scelta del gioco è ricaduta su Tokyo, detto anche 21. Tokyo è un gioco di dadi, che richiede un numero di partecipanti da due in su. Per maggiore semplicià nella grafica abbiamo preferito fissare il numero massimo di partecipanti, e abbiamo deciso di limitarlo a otto.
Lo scopo di ogni giocatore in ogni turno è riuscire a passare al giocatore successivo un punteggio, ottenuto dal lancio di due dadi, sempre maggiore di quello che si è ricevuto dal giocatore precedente, anche bleffando sul punteggio realmente ottenuto. Durante il gioco vengono assegnate penalità, per chi non riesce a passare il punteggio necessario: chi raggiunge le 5 penalità viene eliminato dal gioco. Vince l'ultimo giocatore che rimane.

1.1 Modalità di gioco

Il primo giocatore lancia una coppia di dadi a 6 facce, nascondendo agli altri il risultato del lancio (nella realtà si utilizza un bicchiere dentro il quale tirare i dadi). Guarda il proprio risultato, lo nasconde, e dichiara un punteggio al giocatore dopo di lui. Il punteggio può essere quello reale, o uno diverso, maggiore o minore.
Il giocatore successivo può

- accettare il punteggio. In questo caso tira i dadi per ottenere un punteggio maggiore di quello che ha ricevuto. I dadi precedenti non varranno quindi più, non saranno quindi resi noti a nessuno. Al giocatore che lo segue, indipendemente dal punteggio che avrà realmente fatto, dovrà dichiarare un punteggio più alto di quello che ha ricevuto.

- dubitare il punteggio: significa che il giocatore non crede che il bicchiere nasconda davvero il punteggio dichiarato, quindi alza il bicchiere e lo rivela.

Quando un giocatore dubita un punteggio che ha ricevuto, uno dei due giocatori prende penalità, di cui parleremo a breve. Il gioco riprende col giocatore che ha preso penalità, che ricomincierà una nuova partita tirando i dadi di nuovo (senza vincolo di punteggio minimo, ossia quanto avvenuto fino alla penalità è irrilevante). In nessun caso è possibile passare un punteggio uguale o inferiore a quello ricevuto. Si sempre dichiarare un punteggio maggiore, a meno che non si abbia vincolo di

punteggio minimo (il che accade se si è il primo giocatore, se si è preso penalità, o se si è ricevuto il turno dopo un giocatore che ha abbandonato il gioco).
Il solo caso in cui non si ritira è quando si riceve un 21. In questo caso la scelta è tra dubitarlo o accettarlo senza però ritirarlo. Anche accettando, poi il gioco riprende dall'inizio dal giocatore che ha accettato.

1.2 Calcolo del punteggio

Il punteggio di un lancio di due dadi si ottiene leggendo le cifre sui due dadi come le decine e le unità di un solo numero. Il valore più alto rappresenta le decine, il più basso le unità: ad esempio la coppia (2, 4) corrisponderà al punteggio 42 e non 24. Questi numeri si ordinano tra loro nel normale ordine numerico.
Fanno eccezione le coppie di numeri uguali tra loro (es. 11, 22..): queste si trovano dopo le coppie di numeri diversi (valgono di più), tra loro sempre nell'ordine numerico.
Fa eccezione anche il 21: è il punteggio più alto.
L'ordine risultante da queste regole risulta essere: 31 32 41 42 43 51 52 53 54 61 62 63 64 65 11 22 33 44 55 66 21.
Il 21 non può essere superato da alcun punteggio.

1.3 Penalità

Ci sono due situazioni in cui vengono assegnate penalità: quando un giocatore dubita un punteggio, e quando viene consegnato un 21.
Nel primo caso, se Alba passa un punteggio a Bianca, e Bianca dubita il punteggio, i dadi vengono scoperti. A questo punto

- se il punteggio nascosto è minore di quanto dichiarato da Alba, Alba prende penalità. La penalità à di 2 punti se il punteggio dichiarato era un 21, di 1 punto altrimenti.
- se il punteggio nel bicchiere è maggiore o uguale di quello dichiarato, Bianca prende penalità. La penalità è di 2 punti se il punteggio nascosto è un 21, di un punto altrimenti.

Il secondo caso di penalitá si ha quando un giocatore riceve un 21. Il giocatore può accettarlo (senza scoprire i dadi veri) e riceverà 1 punto di penalità, oppure può dubitarlo. Se lo dubita, si ricade nella regola precedente e il giocatore che dubita prenderà non 1 ma 2 punti di penalità se il 21 era davvero nel bicchiere. Se invece il 21 dichiarato non era vero, sarà il giocatore che lo aveva dichiarato a prendere 2 punti di penalità.

2 Aspetti progettuali

In questa sezione discuteremo delle scelte progettuali fatte per realizzare l'applicazione del gioco in Java, utilizzando RMI come meccanismo di comunicazione tra gli

host, ovvero i giocatori. L'algoritmo per lo svolgimento delle partite e della gestione dei crash è completamente distribuito equamente tra i giocatori e l'unica forma di centralizzazione presente è il server di registrazione.

2.1 Stato condiviso

Per poter giocare la partita, ogni giocatore mantiene le informazioni di tutti i partecipanti al gioco in una lista contenente le seguenti informazioni:

- **identificativo**: contenente informazioni come indirizzo IP e nome del giocatore;
- **stato attuale**: alive o crashed;
- **numero di penalità attuali**: 0-5, se superiore a 5 si tratta di un giocatore eliminato.

Questa lista viene creata dal server al momento della registrazione al gioco e viene comunicata a tutti i giocatori. I giocatori vengono inseriti secondo l'ordine in cui si sono registrati alla partita e l'indice numerico che li indentifica nella lista rappresenta il turno di gioco assegnatogli e di conseguenza l'ordine rispetto agli altri giocatori. Inoltre, ciascuno mantiene l'informazione su quale giocatore ha attualmente il turno, cioè su chi ha i dadi in mano in quel momento.

Queste informazioni vengono aggiornate durante lo svolgimento della partita. Ogni volta che avviene una modifica, viene comunicato l'aggiornamento della singola informazione a tutti i giocatori che provvedono a modificare la loro copia dello stato condiviso.

2.2 Topologia della rete e comunicazioni

Date le informazioni viste in precedenza ogni giocatore è a conoscenza di tutte le informazioni necessarie a connettersi a ciascuno degli altri giocatori e può comunicare con ciascuno di essi. Infatti per le comunicazioni di notifica di un cambiamento di stato, l'informazione viene comunicata dal giocatore che modifica lo stato a tutti gli altri con un broadcast.

Gli stati vengono modificati in base alle regole del gioco, durante lo svolgimento della partita, queste modifiche possono essere:

- assegnazione di una penalità a un giocatore;
- passaggio del turno ad un altro giocatore;
- cambiamento dello stato di un giocatore a "crashed".

In quest'ultimo caso a notificare il crash di un giocatore è quello precedente ad esso, in quanto ogni giocatore controlla costantemente se il successivo è ancora vivo, creando una rete ad anello unidirezionale. Nel caso si accorga del crash lo comunica

a tutti gli altri giocatori e passa a controllare il successivo ancora vivo.

Oltre alle comunicazioni generali di notifica di stato, per giocare la partita i giocatori si passano i dadi in una comunicazione one-to-one. Il giocatore che ha il turno tira i dadi e passa al successivo il valore dichiarato e il reale punteggio ottenuto, nascosto. Il giocatore che riceve i dadi potrà vedere inizialmente solo il punteggio dichiarato. Se decide di dubitare il punteggio reale verrà svelato e sarà calcolata la penalità da assegnare.

Messaggi testuali vengono mandati a tutti i giocatori esclusi dalla comunicazione one-to-one in corso in quel momento, per mantenerli informati su cosa sta accadendo nel frattempo, nonostante queste informazioni non siano necessarie per il corretto svolgimento del gioco (lo "storico dei tiri" non è rilevante, il gioco non ha memoria dei tiri) ma siano piuttosto significative per rendere il giocatore maggiormente partecipe.

2.3 Gestione dei crash

Durante lo svolgimento ordinario del gioco, cioè in assenza di crash, il leader è il giocatore di turno, che tira i dadi o dubita, determinando chi sarà il prossimo giocatore di turno. Come già detto in precedenza la rilevazione dei crash avviene tramite una rete ad anello in cui ciascuno controlla il giocatore successivo. Se un giocatore si accorge di un crash automaticamente diventa il leader del gioco, ovvero si occupa di gestire il turno nel caso in cui il giocatore crashato detenesse il turno di gioco, e lo assegna al primo giocatore vivo successivo a quello crashato.

2.4 Termine del Gioco

Nel corso della partita può verificarsi che diversi giocatori crashino o escano dal gioco per aver raggiunto il massimo delle penalità. Il gioco quindi può terminare quando rimane un unico giocatore vivo e che non ha raggiunto il massimo delle penalità. Questo sarà il vincitore del gioco. Può anche accadere che non ci sia nessun giocatore vincitore nel caso in cui tutti i giocatori crashino e l'ultimo rimasto vivo abbia superato il massimo delle penalità.

3 Aspetti Implementativi

3.1 Strutturazione del codice

3.1.1 Packages

Il codice è strutturato in 6 differenti packages, per permettere al meglio la divisione dei ruoli e delle classi. Questi package sono:

- **unibo.lsb.communication** - al suo interno ci sono tutte le classi Java che permettono di realizzare le comunicazioni tra i vari processi e nella fase di registrazione al server.

- **unibo.lsb.exception** - contiene 2 classi, una che estende l'eccezione base per aggiungere alcune utili informazioni per il progetto, e un'altra che contiene alcune costanti utilizzate all'interno del codice.
- **unibo.lsb.graphic** - questo package contiene tutte le classi che permettono di creare la grafica o che vengono utilizzate dalle finestre per fornire informazioni sullo stato di avanzamento del gioco.
- **unibo.lsb.logic** - in questo package troviamo tutte le classi che realizzano la logica vera e propria del gioco, dunque le regole, gli oggetti che vengono utilizzati durante una partita e le classi che permettono di realizzare i giocatori.
- **unibo.lsb.server** - in questo package ci sono le classi che realizzano l'unica parte centralizzata del progetto, ovvero il server di registrazione.
- **unibo.lsb.tokyo** - questo package è composto da un'unica classe, il cui scopo è quello di far partire effettivamente il gioco aprendo la finestra principale.

3.2 Scelte implementative

In fase di progettazione, e quindi successivamente in fase di sviluppo, abbiamo deciso di fare un uso abbondante ma mirato del design pattern Singleton, questo poichè il progetto include alcuni oggetti che devono essere acceduti da diverse parti del codice senza dover essere ricreati tutte le volte, in quanto contengono informazioni importanti valide per tutta la durata del gioco.
Alcuni esempi di questo utilizzo sono:

- **unibo.lsb.logic.PlayerSelfSingleton** - lo scopo di questa classe è quello di mantenere un'istanza comune e unica che contenga tutte le informazioni sul giocatore, come ad esempio il nome all'interno del gioco, l'IP dal quale si sta giocano e l'avatar attraverso il quale si è riconosciuti durante il gioco stesso.
- **unibo.lsb.graphic.GameTableWindowSingleton** - questa classe mantiene l'istanza della finestra principale di gioco. Tutti gli aggiornamenti grafici, come ad esempio l'aggiunta di penalità, la rimozione di giocatori, il passaggio di turno, vengono notificati all'utente usando questa classe per reperire la finestra attualmente visualizzata, e agendo poi su di essa in modi diversi a seconda della notifica da eseguire.
- **unibo.lsb.communication.InformationSingleton** - questa classe mantiene un'istanza unica e comune di due importanti oggetti del nostro gioco, che sono:
  - Game
  - PlayersTable

  L'oggetto di tipo Game contiene tutte le informazioni sul gioco prima che questo sia iniziato, come ad esempio la lista dei giocatori e il nome del gioco

stesso, mentre invece la PlayersTable contiene le informazioni sul gioco in corso, quindi il numero di penalità di ogni giocatore, il turno corrente e così via.

3.3 Diagrammi

3.3.1 Diagramma delle classi

3.3.2 Diagramma delle interazioni

In figura 1 è mostrato il flusso di operazioni svolte quando si riceve un'informazione sullo stato di un giocatore (tipicamente, che questo è crashato). Sostanzialmente, vengono aggiornati l'oggetto PlayersTable, e l'oggetto che costituisce la finestra grafica. Entrambi gli oggetti vengono ottenuti accendendo a relativo oggetto singleton (abbiamo mostrato esplicitamente l'accesso al singleton solo per il primo oggetto). Seguono lo schema di questo metodo remoto, molto semplice, altri metodi remoti quali il metodo deputato a riceve messaggi testuali, e quello che riceve l'informazione su chi sia il giocatore che ha vinto.
In figura 2 sono riassunte le azioni nel caso di ricezione di un'informazione relativa al turno. Il giocatore aggiorna il suo stato con la nuova informazione, e se il turno è passato a lui gioca e manda il risultato della sua giocata a successivo nodo vivo nell'anello, notificando allo stesso tempo l'accaduto agli altri nodi con un messaggio testuale. Le comunicazioni in uscita sono gestite tutte dalla classe Communication-Local.
Un giocatore riceve un turno che corrisponde al suo solo se il giocatore prima di lui è crashato o ha terminato le sue penalità ed è quindi uscito dal gioco. Se viene ricevuto un turno, si tirano i dadi senza vincolo di punteggio minimo, quindi è legittimo aspettarsi ci sia senza eccezioni un nuovo dado da mandare al nodo successivo.
In figura 3 è rappresentata la risposta alla ricezione di una penalità. Se il giocatore che riceve la penalità è lo stesso al quale la penalità è rivolta, questi non si limita ad aggiornare le proprie informazioni. Se con la nuova penalità il giocatore è escluso dal gioco, passa il turno al primo giocatore vivo dopo di lui, notificando a tutti i partecipanti al gioco(lui compreso) il nuovo detentore del turno. Se invece è ancora in gioco, notifica agli altri giocatori che detiene il turno, gioca, e manda i dadi al successivo giocatore vivo.
Infine, la figura 4 riassume la logica i ciò che avviene quando vengono ricevuti dei dadi. Abbiamo omesso di rappresentare le costanti chiamate alla grafica e gli invii di messaggi di testo agli altri giocatori, per rendere il diagramma più leggibile. Chi riceve i dadi ha il turno, e lo notifica agli altri giocatori. Se accetta i dadi, li ritira e manda i nuovi dadi al successivo giocatore vivo. Se accetta un dado dichiarato 21, prima di tutto notifica la penalità che ha ricevuto, poi se è ancora in gioco (non ha raggiunto il numero massimo di penalità) lancia i dadi e li manda, altrimenti notifica agli altri giocatori che il turno è del successivo giocatore vivo. Se invece dubita, notifica agli altri la penalità e chi l'ha presa (se lui o il giocatore che gli ha passato i dadi). Se chi ha preso penalità era il giocatore precedente ma questi è nel frattempo crashato, il giocatore corrente ha il turno e lo notifica a tutti compreso se

stesso. Se invece è il giocatore corrente ad aver preso penalità, se è ancora nel gioco tira i dadi e li manda avanti.

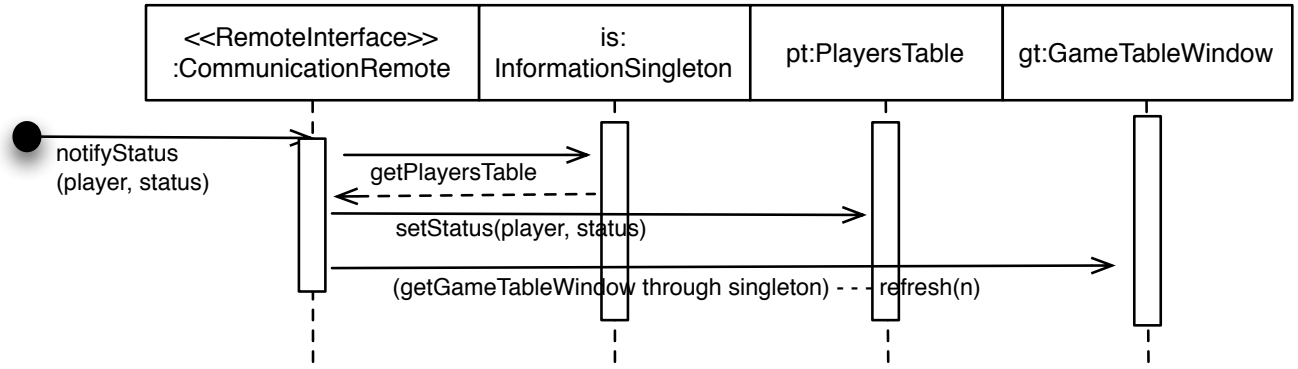

Figura 1: Diagramma di sequenza che rappresenta la ricezione di informazioni sullo stato di un giocatore.

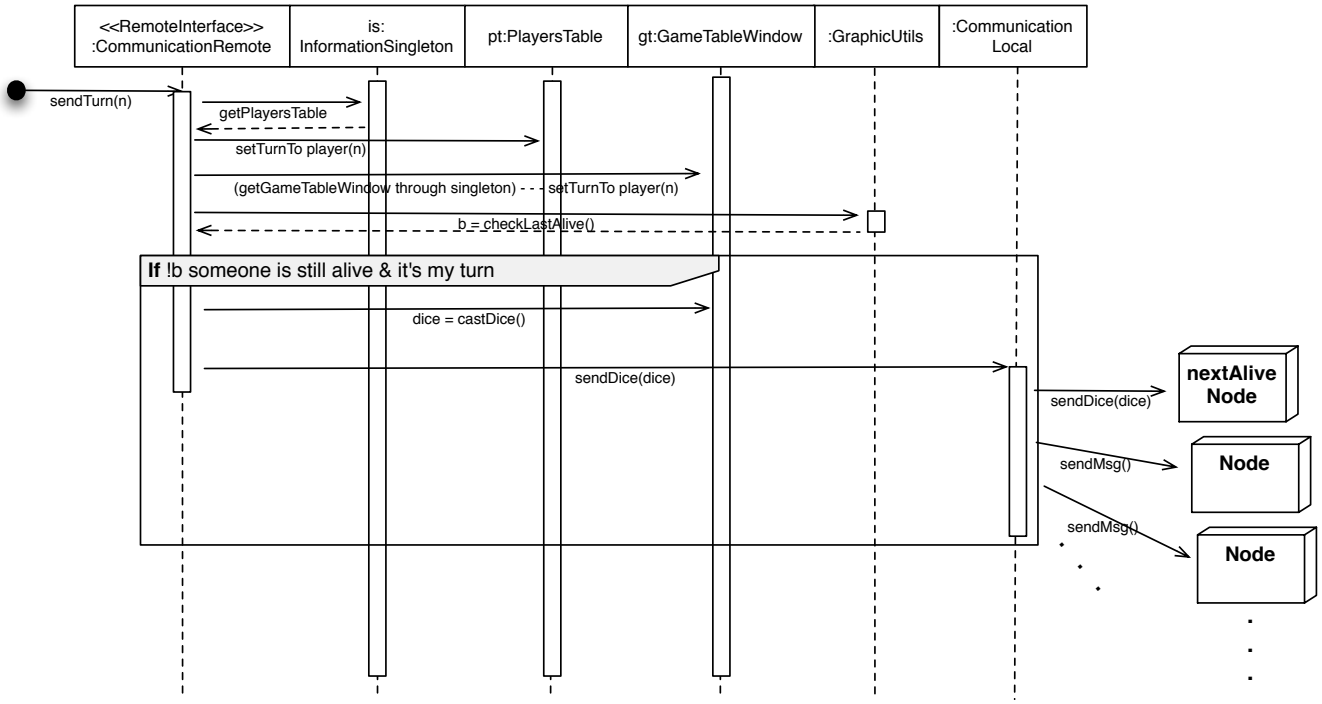

Figura 2: Diagramma di sequenza sulla ricezione di informazioni su chi detiene il turno.

3.4 Reazione ai crash

Il nostro progetto resiste al crash di n-1 giocatori, anche contemporaneo, fino ad eleggere il vincitore.

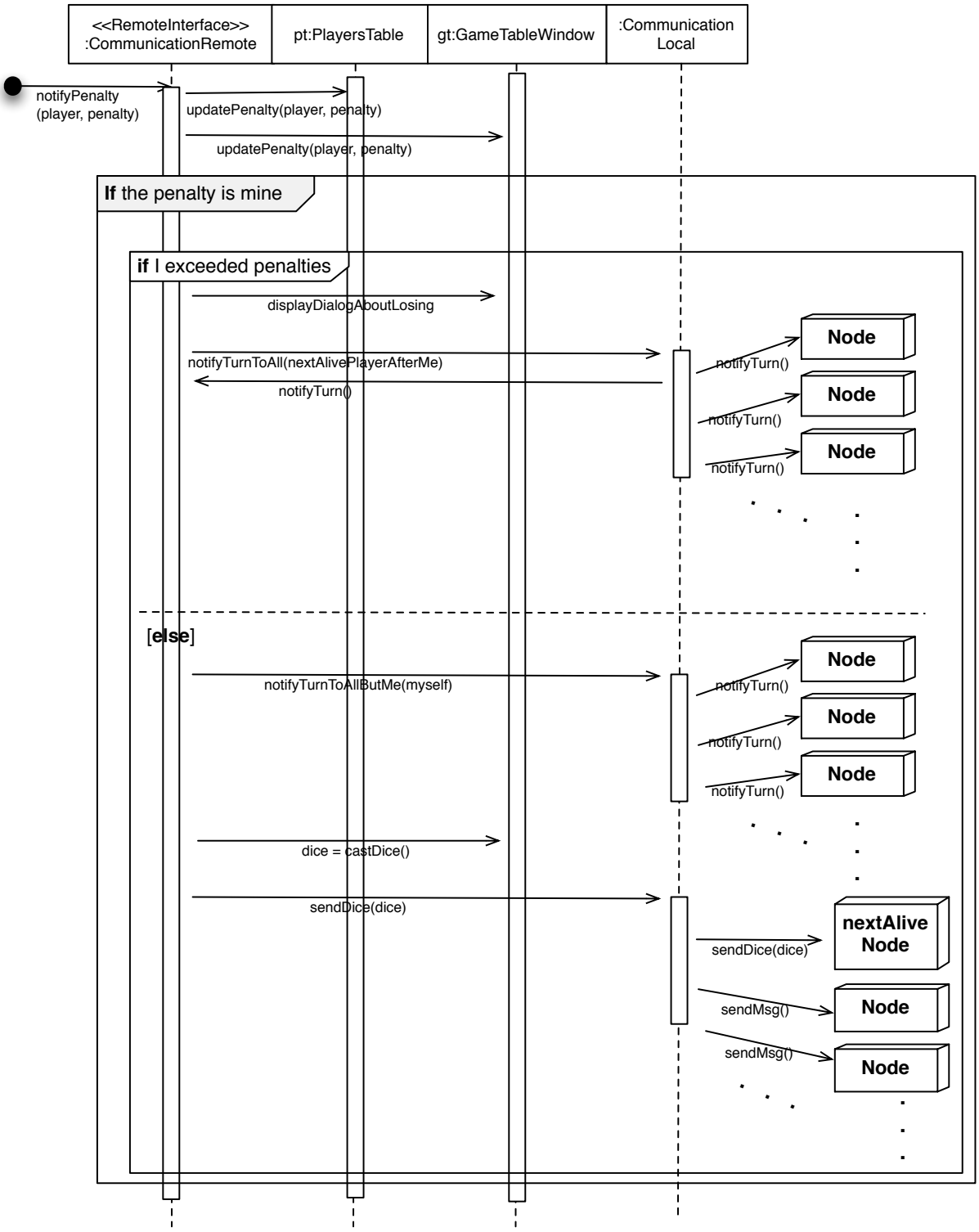

Figura 3: Diagramma di sequenza relativo alla ricezione di una penalità.

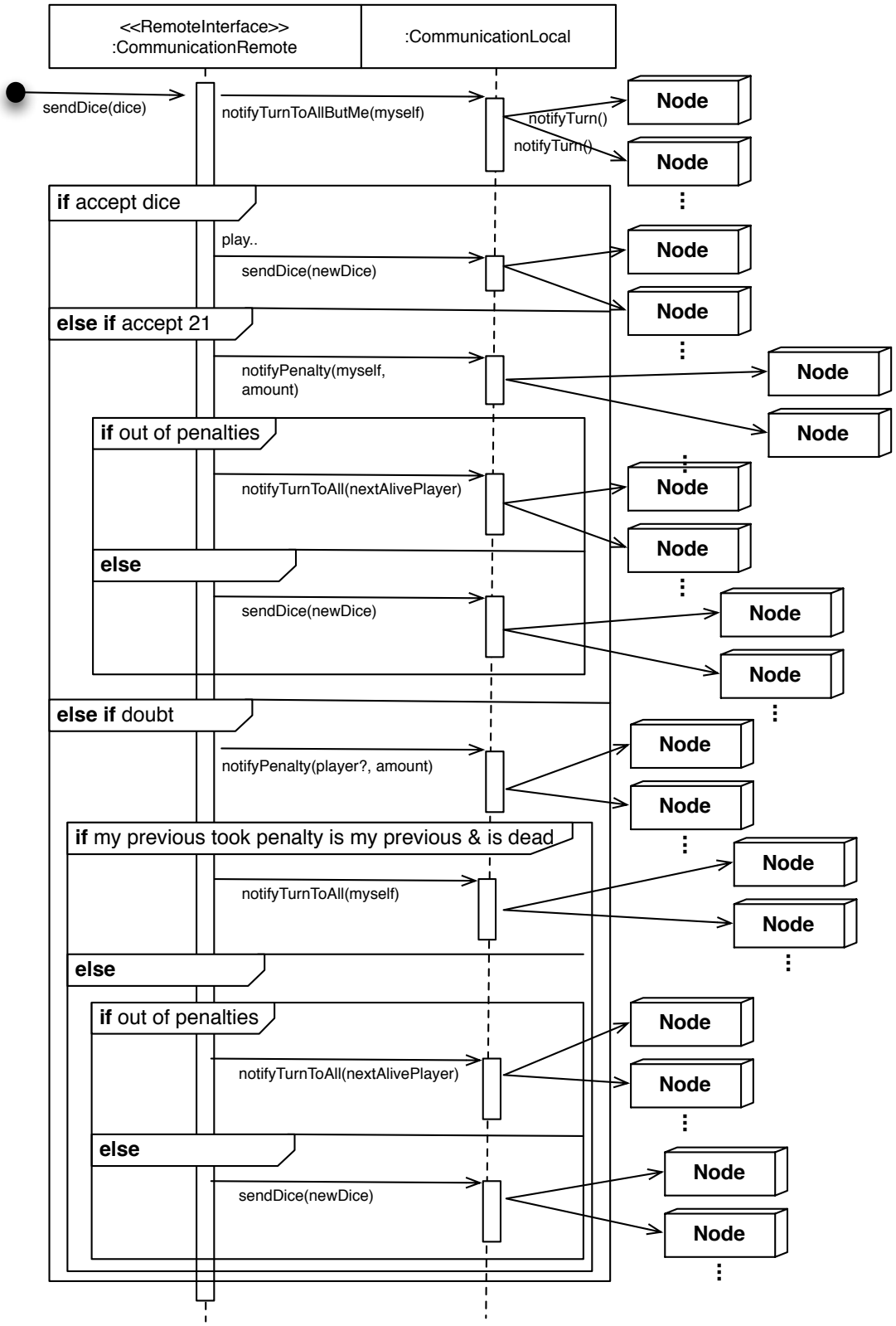

Figura 4: Diagramma di sequenza che rappresenta la ricezione dei dadi.

Abbiamo cercato di creare processi il più possibile paritari fra loro, infatti il nostro leader è sempre colui che ha il turno attuale.
Quando parte il gioco, ogni giocatore fa partire un polling verso il giocatore immediatamente successivo a se stesso. Ogni 0,2 secondi controlla che questo giocatore sia vivo e che non abbia superato il numero massimo di penalit‘a, e in caso negativo procede a compiere alcune operazioni descritte di seguito.
Tutte le volte che un giocatore viene trovato in stato di crash o eliminato, viene chiamato il metodo *reactOnDeadPlayer*, che si preoccupa di notificare il nuovo stato del giocatore a tutti i partecipanti, e nel caso in cui che quel giocatore fosse il leader, anche il turno.
Abbiamo preferito controllare non solo lo stato di crash, ma anche quello di massima penalità, per il fatto che un giocatore che viene eliminato dal gioco è più propenso a lasciarlo. In questo modo, noi anticipiamo questo possibile comportamento modificando il polling facendolo passare al giocatore successivo.