PirateGoose Game

Progetto di Sistemi Distribuiti

Realizzato da: Fabio Quinzi, Luca Valentini

1 Feb 2014

Sommario

Questo documento descrive la progettazione di un sistema distribuito, nello specifico caso, un gioco da tavola immaginato sulla base del famoso “gioco dell’oca” e chiamato Pirate Goose. Tale sistema dovrà permettere il gioco a 4 players, utilizzerà come tecnologia di comunicazione Java RMI e come da specifiche sostenere 3 guasti di tipo crash.

Introduzione

L’obiettivo del progetto è la creazione di un sistema distribuito su cui implementare un gioco multiplayer, ogni giocatore sarà costituito da un peer e la comunicazione tra di essi dovrà essere garantita anche in caso di crash di alcuni.
I requisiti delle specifiche in dettaglio sono:

1. stato condiviso dai giocatori, che dipende dal tipo di gioco, quale ad esempio il tabellone, il tavolo con le carte giocate, il mazzo, … .
2. linguaggio Java.
3. le comunicazioni sono affidabili, ossia non si guastano.
4. i processi possono avere solo guasti di tipo crash.
5. tolleranza di almeno due guasti, sino al numero di tutti i giocatori tranne il vincitore.
6. numero di giocatori maggiore di 3.
7. RMI e non socket.
8. l'architettura è distribuita, i processi sono tutti pari.
9. l'unica forma di centralizzazione può essere il servizio di registrazione al gioco.

Prendendo d’esempio il famosissimo “gioco dell’oca”, ne è stata realizzata una sua variante chiamata Pirate Goose.
Pirate Goose è un gioco a cui possono sfidarsi 4 giocatori, è basato anch’esso, come il gioco dell’oca, su un tabellone 10x9 (Fig. 1) dove la pedina di ogni singolo giocatore percorre un numero progressivo di caselle fino ad arrivare o superare una casella finale dove è situato il tesoro per poter ottenere la vittoria.
Il percorso è studiato per avere determinate caselle che contengono bonus ed altre che contengono malus, queste caselle sono bilanciate ad eccezione di una casella particolare che fa ripartire il giocatore dall’inizio del tabellone.

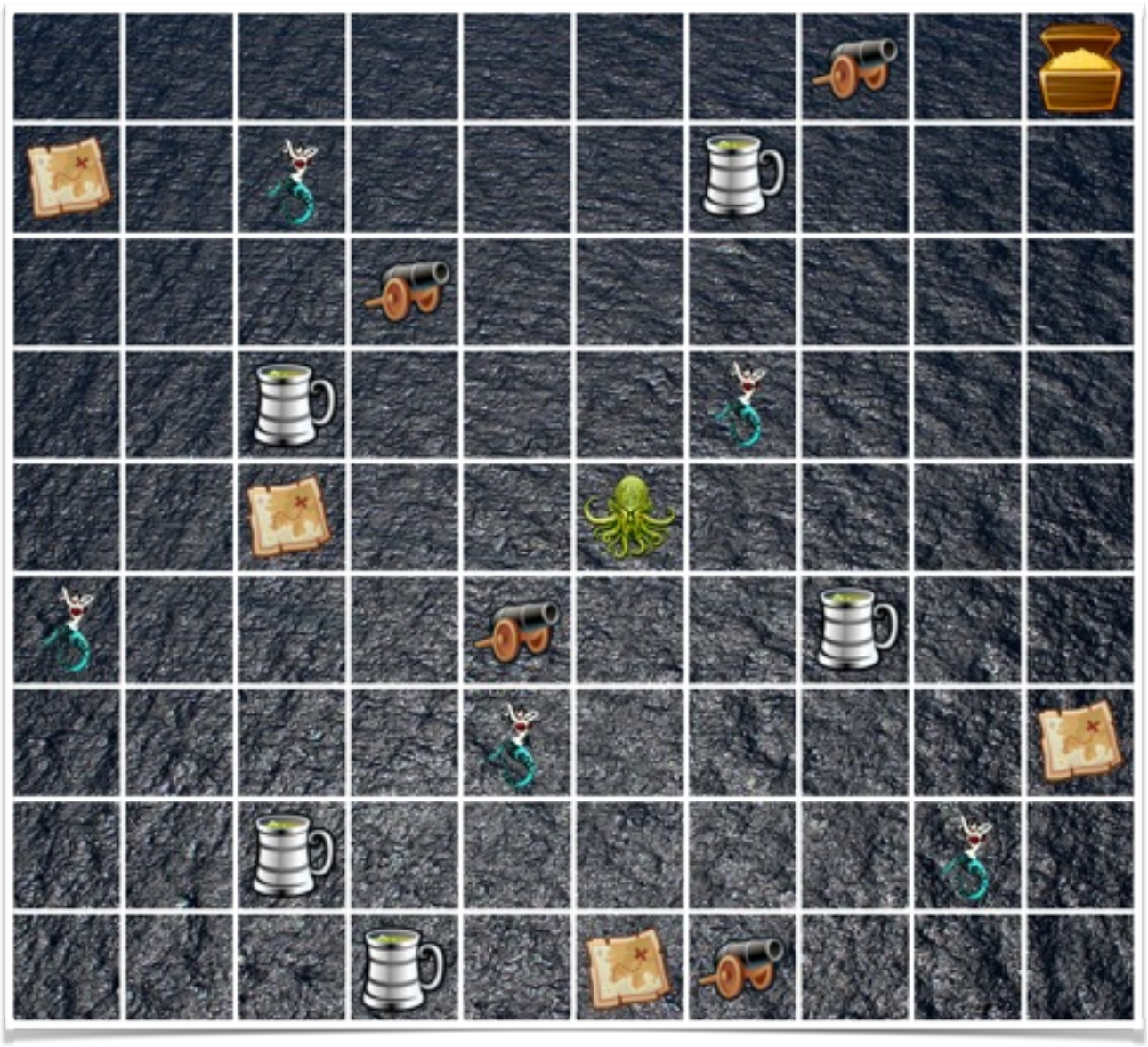

Fig. 1: Tabellone di gioco

Le caselle di bonus e malus consistono in stati di avanzamento e retrocessione rispetto alla posizione attuale, in tutto i tipi di caselle speciali sono 5, ovvero:

Sirena	Cannone	Rum	Mappa del Tesoro	Kraken

Aspetti Progettuali

Come suggerito da specifiche, l'architettura deve essere distribuita e ogni processo deve essere paritario con unica possibile eccezione del servizio di registrazione al gioco, quindi abbiamo deciso di sfruttare il modello di architettura P2P per i giocatori, implementando un server di registrazione come servizio di "lancio" del gioco.
Il primo processo che verrà quindi lanciato sarà il server (PirateGooseServer), che rimarrà in attesa dei 4 giocatori da mettere in comunicazione.
Una volta che un giocatore (PirateGooseClient) si connetterà al server rimarrà in attesa degli altri 3 giocatori e quando il server avrà le richieste di tutti e 4 i giocatori comunicherà in broadcast una lista di indirizzi contente l'IP di ciascuno per far in modo che da li in avanti comunichino tra di loro.
Da questo momento in poi il server rimarrà in attesa di un altro gruppo di giocatori o potrà semplicemente essere terminato dato che non sarà più parte dell'interazione tra i giocatori.

Il server mandando la lista degli indirizzi IP si occupa anche di dare un ordine ai giocatori che inizieranno da questa informazione a gestire i propri turni.
Essendo un gioco a turni l'architettura che più ci è sembrata adatta è stata sicuramente quella ad anello con passaggio di token, questo anche perché può esserci un solo giocatore attivo alla volta (il giocatore con il Token) e una sola direzione nella successione dei turni.
La struttura ad anello con passaggio di token ci è sembrata la scelta migliore anche pensando a dover riorganizzare i giocatori in caso di crash, con questa architettura basta infatti un semplice aggiornamento della struttura della rete per riorganizzare i turni.

Una volta che un giocatore ha il turno per mezzo del token, gli viene concessa la possibilità di lanciare il dado, in base al punteggio si sposterà su una cella di destinazione e poi se questa cella avrà bonus o malus, questi verranno applicati al tabellone modificando lo stato del giocante, finita questa fase la mossa verrà mandata tramite broadcast a tutti gli altri player e il token verrà passato al giocatore successivo.

Lo stato condiviso viene mantenuto dal tabellone, aggiornato ogni fine turno da tutti i giocatori, in modo da evitare perdite di dati nel caso di crash e in modo da garantire che un

giocatore possa aggiornare la propria visione globale dopo ogni turno non dovendo aspettare i turni degli altri giocatori con lo stesso immutato stato di gioco.

Aspetti Implementativi

ARCHITETTURA

Ogni giocatore è rappresentato da un nodo di una rete totalmente connessa dove il turno attivo (ovvero il nostro token) viene gestito in ricezione.
Si è scelta una rete totalmente connessa rispetto ad una rete ad anello per semplicità nella gestione dei messaggi in caso di crash, in questo modo ogni giocatore avrà sempre lo stato globale aggiornato e non dovrà aspettare il suo turno per avere lo stato aggiornato.

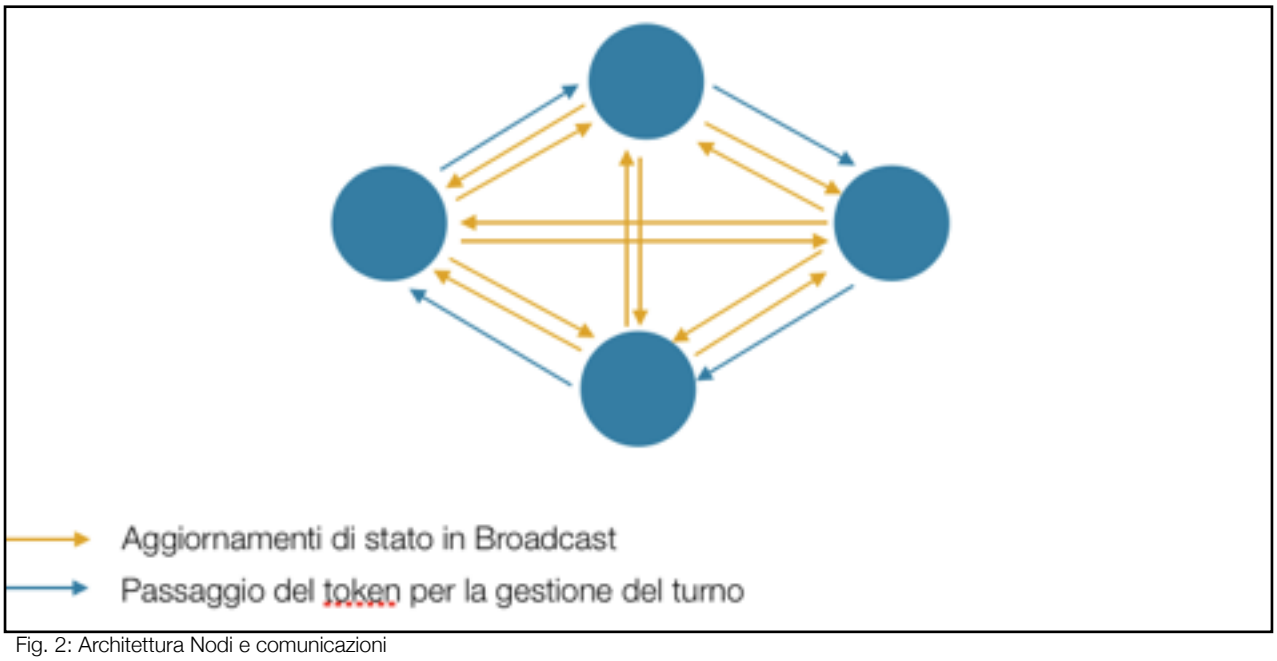

Fig. 2: Architettura Nodi e comunicazioni

CONNESSIONE E ATTESA PARTITA

La registrazione di un giocatore alla partita viene effettuata da una chiamata RMI sul server registry, un server sempre attivo che aspetta richieste da parte di giocatori e fa in modo che ci siano sempre 4 giocatori per ogni partita. Connettendosi a questo server ci si mette in attesa, l'attesa termina quando il server ha ottenuto le richieste di 4 player e invia una risposta a tutti contenente una lista di indirizzi, questo è effettuato tramite una HashMap dove sono contenute le seguenti informazioni: Chiave HashMap, IP, ID Giocatore.

Con queste informazioni un giocatore è poi in grado di calcolarsi l'URI RMI per comunicare con gli avversari.
Ogni player prima di ricevere la risposta dal server registry ha intanto ha creato il proprio servizio RMI per ricevere comunicazioni, appena arrivata la risposta dal server tutti i giocatori creano la propria lista di player e controllano se i relativi servizi di comunicazione RMI sono attivi.
Il primo a giocare è il primo in ordine di ID nella HashMap spedita dal server ancora connesso, se un giocatore durante la fase di attesa degli altri player si disconnette, il gioco inizierà comunque con un numero inferiore di partecipanti.

Fig. 3: Visualizzazione attesa altri Giocatori

MOSSA E PASSAGGIO DI TURNO

L'inizio del turno di un giocatore consiste nell'attesa che il token assuma il valore corrispondente al proprio ID, quando questo avviene, il client attiva la possibilità al giocatore di lanciare il dado.
Quando il dado verrà lanciato, il risultato del dado andrà a modificare lo stato della pedina sul tabellone di gioco, aggiornata la propria interfaccia la mossa verrà passata in broadcast

a tutti gli altri giocatori in modo da consentire l'aggiornamento del tabellone e dello stato condiviso a tutti i partecipanti.
Nello stesso messaggio della mossa verrà anche indicato l'id del player a cui spetterà giocare, in modo che ognuno singolarmente aggiorni questa variabile nel proprio stato locale e controlli se dare o meno la possibilità di lanciare il dado al giocatore.
Ultimo dato importante incluso nel messaggio di broadcast della mossa è lo stato del match che permette di capire se c'è un vincitore ed la partita si è conclusa.
Il metodo chiamato sui nodi per far in modo che ricevano il messaggio aggiornamento dello stato globale è il seguente:

```
public void receiveMessage(int senderID, int receivedToken, int moveValue, boolean gameEnded)
```

GESTIONE CRASH

La gestione degli eventuali crash è effettuata da un metodo RMI chiamato Ping.
Questo metodo permette ad ogni nodo di controllare se il precedente, nella sequenza di giocatori, è ancora vivo o meno. Ogni client controlla esclusivamente lo stato del suo nodo precedente.

```
public boolean receivedPing(int clientID);
```

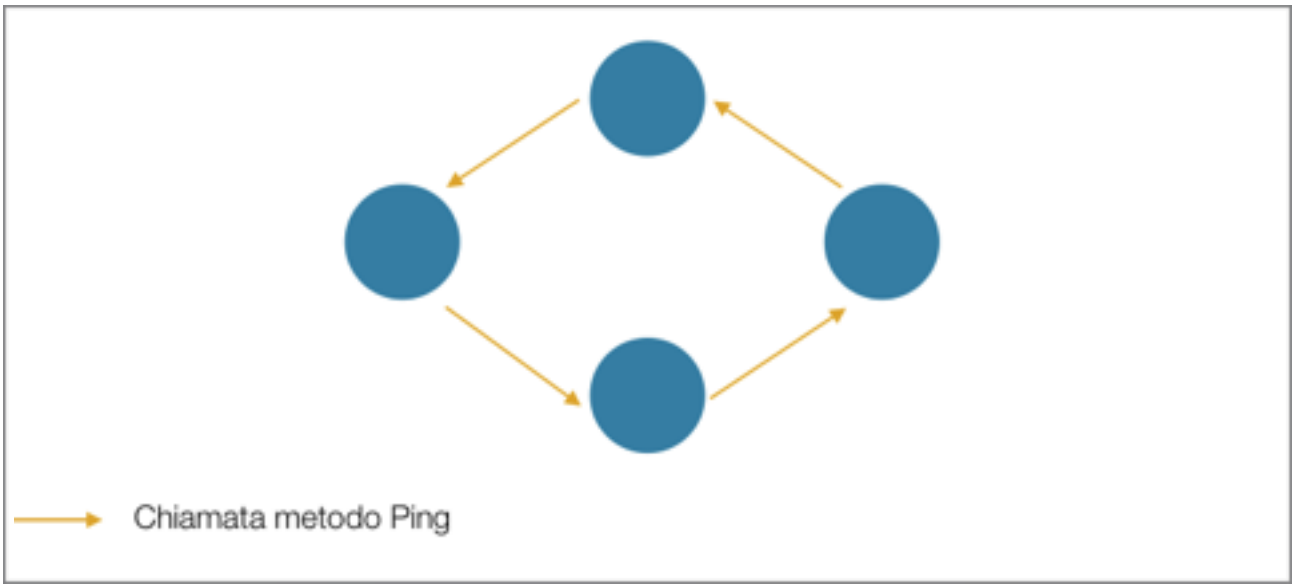

Fig. 4: Direzione chiamata metodo Ping

Quando si verifica un crash, ovvero la chiamata del metodo Ping effettuata da un player solleva un'eccezione, il nodo non raggiungibile viene rimosso dalla lista dei nodi giocanti e il crash viene poi comunicato a tutti gli altri giocatori attivi di modo che possano anche loro aggiornare la propria struttura di rete tramite il metodo removeClient:

public void removeClient(MatchHandlerInterface deadClient)

Dato che ogni client conosce lo stato del token precedente all'evento di crash, calcolare a

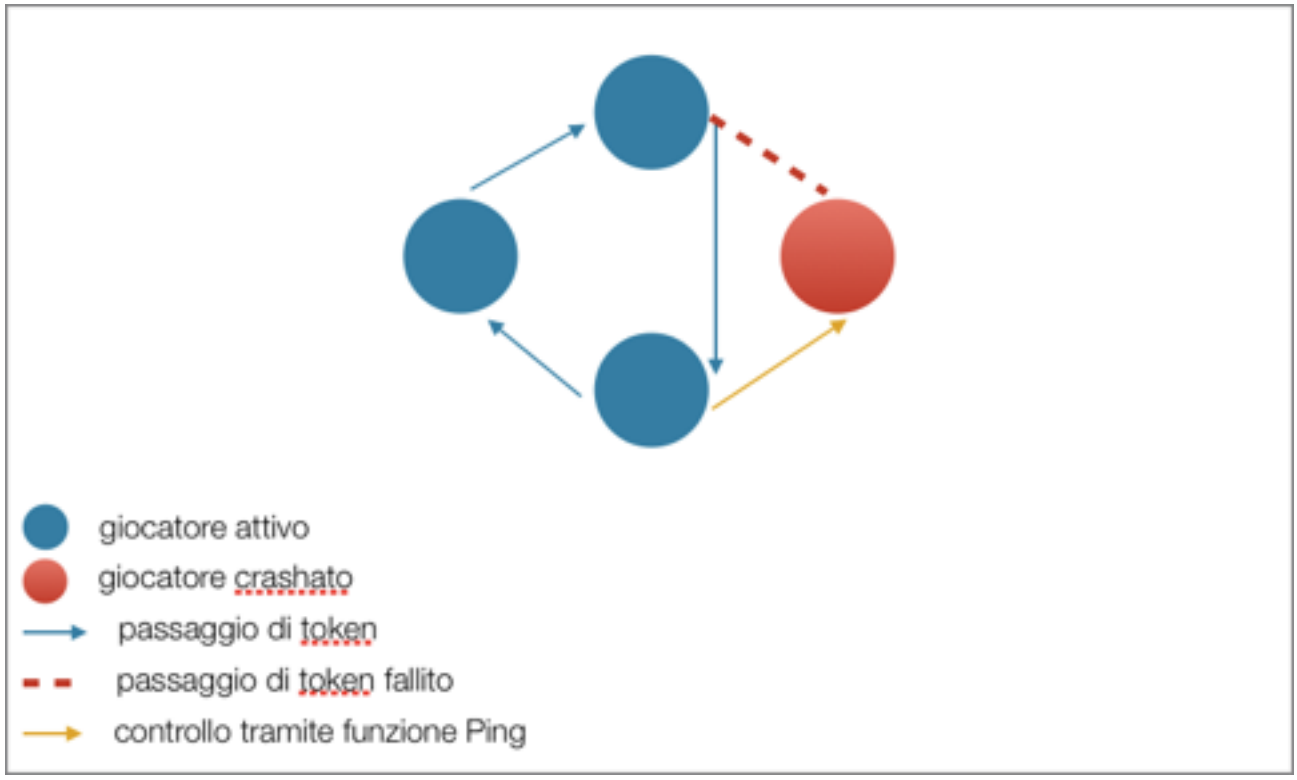

Fig. 5: Riorganizzazione della rete in risposta al crash di un nodo giocante

chi andrà il token sarà un'operazione banale effettuata sulla nuova lista di giocatori in cui non sarà presente il player crashato.

GESTIONE UI

Per la gestione dell'interfaccia grafica abbiamo deciso di utilizzare le librerie JGameGrid, la scelta è dovuta al fatto che abbiamo già utilizzato le stesse librerie per altri progetti e

conoscevamo già bene come utilizzarle, inoltre rendono rapido lo sviluppo di un gioco dalla semplice grafica come quello che abbiamo creato e sono molto affidabili.
La comunicazione tra nodo e interfaccia viene gestita dalle due classi MatchHandler e GameView, queste gestiscono il passaggio di informazioni garantendo l'aggiornamento dell'interfaccia dopo ogni mossa.

Fig. 6: Interfaccia di gioco durante una partita

Valutazione

Le specifiche scelte adottate sono state il risultato di vari test e valutazioni che abbiamo effettuato durante la progettazione e sviluppo del progetto, per quanto riguarda la l'operazione di rilevazione del crash, questa poteva essere effettuata per esempio dal giocatore che si trovava a passare il token, da quello che l'avrebbe dovuto ricevere o da tutti i giocatori, quello che abbiamo scelto, ovvero che ogni giocatore controlli solo ed esclusivamente il precedente nodo nella lista dei giocanti, è stato per evitare ridondanza inutile nei controlli ed evitare attese troppo lunghe al giocatore.

La cosa che ci è sembrata più logica è stata far gestire il controllo ai giocatori inattivi dando un senso al loro stato di attesa.
Il messaggio di aggiornamento dello stato globale, ovvero di aggiornamento del tabellone, viene mandato in broadcast per evitare attese dovute alla rilevazione di crash o in caso di crash del giocatore portatore delle informazioni di stato globale, così ogni giocatore, indipendentemente dai guasti, riceverà un messaggio con lo stato globale aggiornato, il gioco è reso quindi dinamico e non si deve ogni volta aspettare il proprio turno per avere un aggiornamento di interfaccia con le posizioni degli avversari.
La praticità del broadcast per mantenere lo stato globale del gioco sempre aggiornato spiega la nostra scelta di una architettura a rete totalmente connessa invece di una semplice rete ad anello.

Conclusioni

L'obiettivo del progetto era quello di costruire un sistema distribuito su cui implementare un gioco multiplayer, tale sistema è stato sviluppato secondo i requisiti richiesti ed è stato testato per assicurarci del corretto funzionamento.