Bajnarola

Progetto di sistemi distribuiti

Davide Berardi 0000712698 | Matteo Martelli 0000702472 | Marco Melletti 0000699715

2 dicembre 2015

Sommario

Bajnarola e' un bel gioco

1 Introduzione

Sempre più giochi da tavolo ormai vantano una versione digitale, caratterizzata dalla possibilità di permettere agli utenti di giocare in modalità multiplayer. Allo stesso tempo però sono ancora rare implementazioni distribuite di giochi multiplayer le quali spesso sono invece basate su un architettura client server. In questo documento descriveremo il nostro lavoro di progettazione ed implementazione della versione software di un gioco da tavolo con particolare interesse riguardo l'architettura di rete distribuita utilizzata nella modalità multiplayer.

1.1 Carcassone

In particolare è stato nostro interesse occuparci del remake del gioco da tavolo Carcassonne. Quest'ultimo nello specifico è un boardgame dei primi anni 2000 basato su tessere che consiste nel creare un paesaggio medievale posizionando e accostando tra loro vari tipi di tessere, che rappresentano una parte di città, un tratto di strada, un campo o un monastero [1]. Completando quindi più città, strade, ecc, attraverso tali tessere, i giocatori (previsti da 2 a 5) accumulano i punti necessari a vincere la partita. Al fine di comprendere al meglio le prossime sezione di questo documento, vedremo brevemente di seguito il funzionamento del gioco. All'inizio della partita, una singola tessera è posizionata sul tavolo, scoperta; le altre tessere sono posizionate coperte nel mazzo e mischiate. Ciascuna di tali tessere rappresenta un frammento di paesaggio, e può contenere uno o più dei seguenti elementi:

- tratti di strada, inclusi incroci e curve

[1] Si fa riferimento alla prima versione del gioco in cui non sono presenti fiumi

- aree cittadine racchiuse da mura
- campi che circondano le città e accolgono le strade
- un monastero

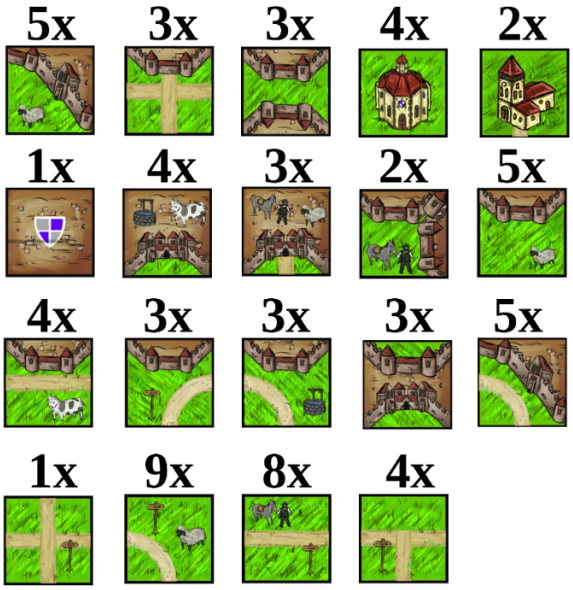


Figura 1: Elenco delle tessere presenti nel gioco

A turno, i giocatori estraggono una tessera dal mazzo e la posizionano scoperta sul tavolo in contatto con le tessere già piazzate attraverso uno o più lati in modo coerente con le altre, in modo da proseguire eventuali strade, campi, o mura già presenti.

Dopo aver posizionato la tessera, il giocatore può decidere di piazzare una pedina detta *meeple* su di essa che reclama la proprietà di un elemento di terreno e non può essere piazzato su un elemento già reclamato da un altro meeple. Può accadere comunque che un elemento possegga più di un proprietario se diviene una congiunzione di due elementi dello stesso tipo non precedentemente adiacenti. Quando un elemento viene completato, se ad esempio le mura di una città

vengono chiuse o se una strada ha due estremità chiuse, il proprietario di quell'elemento acquisisce i relativi punti. Il punteggio di un elemento è dipendente dal numero e dal tipo di tessere che compongono l'elemento.

Il gioco termina con il piazzamento dell'ultima tessera; vince il giocatore che ha totalizzato più punti.

Rimandiamo alla documentazione ufficiale di Carcassonne per ulteriori dettagli.

La semplice struttura del gioco e la sua organizzazione a turni rende interessante l'approccio distribuito in quanto si evita di incorrere in problemi di prestazioni tipici dei giochi reattivi. Quest'ultimi infatti hanno requisiti prestazionali molto stringenti. In

particolare spesso richiedono che le comunicazioni di rete siano a bassa latenza al fine di impedire il più possibile agli utenti di giocare con artefatti grafici fastidiosi dovuti ai ritardi di rete o all'overhead di comunicazione.
Come già accennato, il gioco da noi considerato necessita invece di requisiti prestazionali più laschi soprattuto per i lunghi tempi decisionali degli utenti (comparati ai tempi di rete e computazionali).
Vedremo nelle prossime sezioni i dettagli progettuali ed implementativi. In ultimo forniremo una analisi valutativa dei risultati ottenuti dai test con le dovute considerazioni finali.

2 Aspetti progettuali

Il gioco in questione è stato progettato nell'ottica di un sistema resistente ai guasti, come richiesto da progetto.
L'aspetto critico studiato maggiormente per raggiungere tale obiettivo è stato senz'altro la comunicazione, pensata per risultare robusta e allo stesso tempo in grado di fornire reattività ai client di gioco, nonostante quest'ultimo non utilizzi uno schema realtime.

2.1 Il gioco

2.2 La comunicazione

3 Aspetti implementativi

3.1 Il desing pattern MVC

Il sistema e' stato implementato utilizzando il design pattern **MVC**, Model View Controller, questo design pattern aiuta lo sviluppo di applicazioni "a camere stagne", separando il modello (la logica e il motore di gioco) dalla grafica e dalla sua presentazione all'utente tramite un'entita' denominata **controller**, nello scenario del progetto questa entita' e' stata dotata della logica di gestione distribuita dello stato dei vari mondi di gioco.

3.2 Implementazione dello schema di gioco

In questo paragrafo vedremo brevemente l'implementazione dello schema e della logica di gioco. La figura **??** mostra il diagramma delle classi del componente model il quale racchiude tutte le funzionalità della logica di gioco.
Nello specifico il model contiene tutte le funzionalità che permettono il collegamento logico delle tessere fra di loro. Inoltre la logica implementa i meccanismi per rilevare il completamento degli elementi (città, strade, monasteri) ed assegnare i punti ai loro rispettivi proprietari.

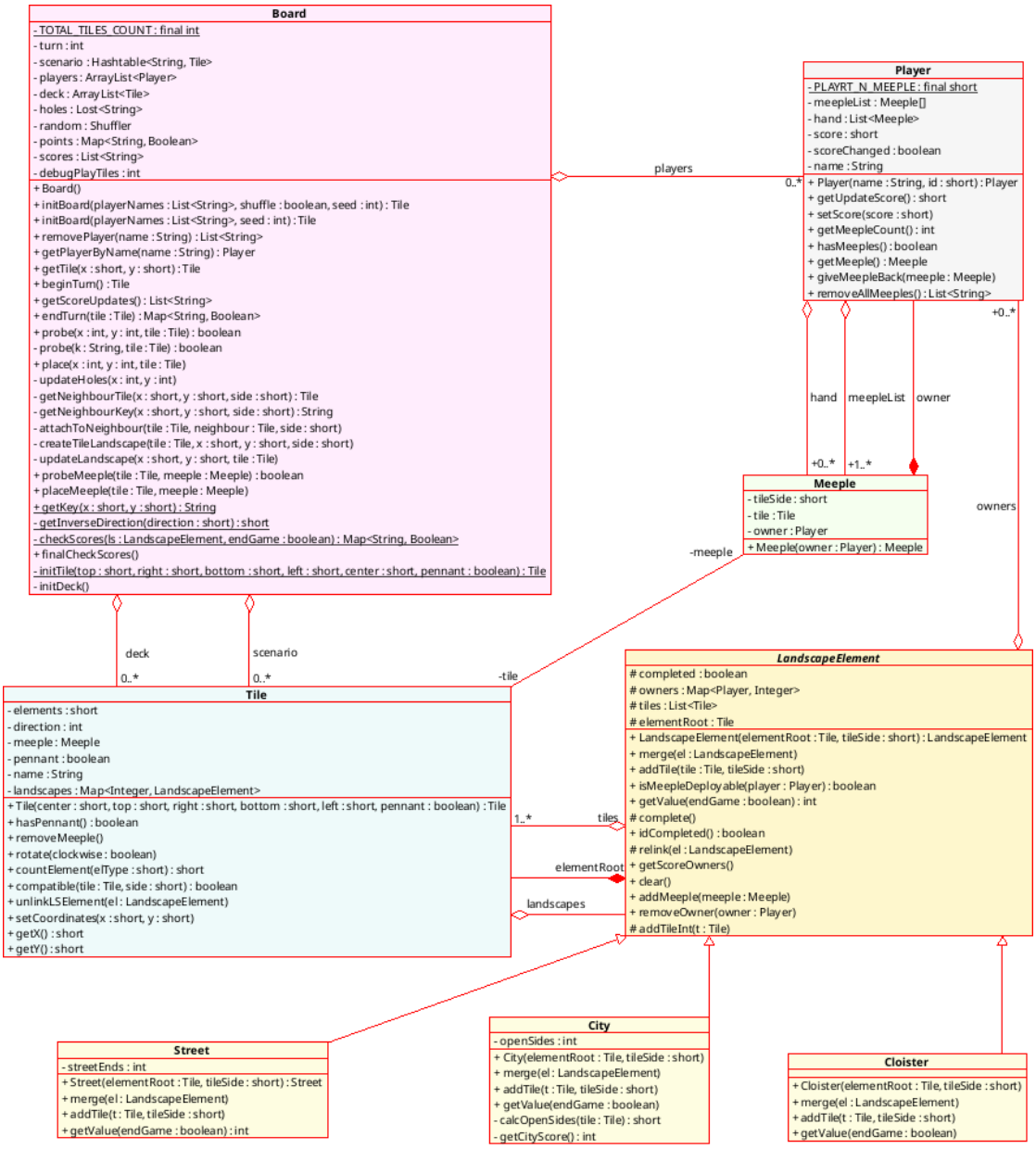
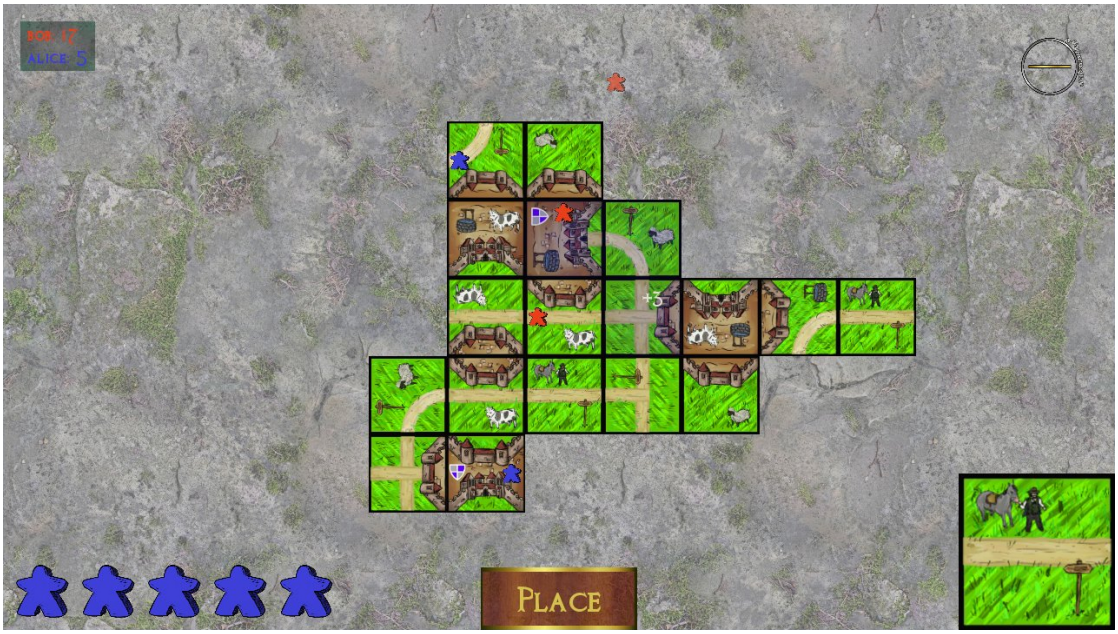


Figura 2: Diagramma delle classi del modello di gioco



3.3 L'interfaccia grafica

L'interfaccia grafica e' basata sul motore Slick2D**??**, questo mette a disposizione una libreria abbastanza semplice per realizzare interfacce grafice anche complesse prive di rendering 3D.

L'intera esecuzione dell'inferfaccia grafica e' affidata ad un thread dedicato, una classe principale gestisce il dispatching degli aggiornamenti grafici e la lettura dell'input.

Il motore di gioco lancia la funzione di rendering alla frequenza impostata, la classe principale delega dunque la composizione della schermata alla scena attualmente caricata. In parallelo viene anche lanciata (con cadenza meno regolare) una funzione di aggiornamento (`void update()`) che interroga il controller e recupera nel thread della grafica gli aggiornamenti logici pendenti, se ce ne sono.

Alla stessa maniera, la gestion degli input viene passata alla scena corrente dalla classe principale in modo da interpretare le pressioni dei tasti e i movimenti del mouse nella maniera corretta.

3.3.1 Interazione

In figura **??** e' mostrata una schermata di gioco, la parte centrale e' occupata dal tabellone su cui vengono piazzate le tessere ed i meeple. Sui contorni della visuale di gioco vengono mostrati gli elementi dell'HUD (*Heads-Up Display*):

- in alto a sinistra i punteggi attuali dei partecipanti ancora in gioco;
- in alto a destra compare un pulsante per attivare e disattivare lo zoom quando lo scenario eccede i limiti dell'HUD stesso, in questo caso si puo' spostare la visuale

muovendo il puntatore verso i bordi della schermata oppure utilizando le frecce direzionali sulla tastiera;

- in basso a sinistra i meeple ancora in mano al giocatore.

Inoltre se un giocatore e‘ di turno l'HUD conterra‘ anche:

- in basso al centro il pulsante di confema del piazzamento;
- in basso a destra la tessera da piazzare;
- *una volta confermato il piazzamento* sulla tessera da piazzare vengono evidenziate le posizioni su cui e‘ possibile mettere un meeple (ammesso che il giocatore ancora ne abbia in mano).

Il giocatore di turno, spostando il mouse sui confini dello scenario creato, puo vedere un contorno nero che indica la possibilita‘ di tentare un piazzamento. Un click sinistro del mouse effettuera‘ il tentativo, mostrando la tessera in trasparenza se il piazzamento e‘ possibile o un effetto rosso in caso contrario.

Utilizzando la rotellina del mouse o il click destro e‘ possibile ruotare la tessera.

Una volta confermato il piazzamento si puo‘ decidere di mettere un meeple su un elemento del territorio della tessera appena piazzata cliccando sul contorno del meeple e lo si puo‘ rimuovere cliccando sul meeple stesso.

Al termine di un turno ogni giocatore vedra‘ i risultati del piazzamento effettuato, gli elementi del territorio completati vengono evidenziati e se qualche giocatore ha segnato dei punti da questi completamenti il punteggio dei singoli elementi viene mostrato sopra agli elementi stessi, come anche il piazzamento o la restituzione dei meeple.

3.4 La gestione e la distribuzione della rete

3.4.1 Registrazione presso la lobby

Il primo passo svolto da ogni nodo di rete e' la registrazione presso un server centralizzato comune, implementante diverse "stanze" di gioco; le cosiddette lobby.
Questo server aspettera' quindi la registrazione del numero specificato di partecipanti, inviando loro la lista dei giocatori per poi lasciare il pieno controllo ad essi, chiudendo la stanza.

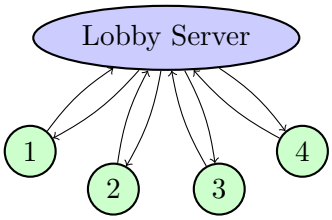


Figura 3: Registrazione presso il server lobby

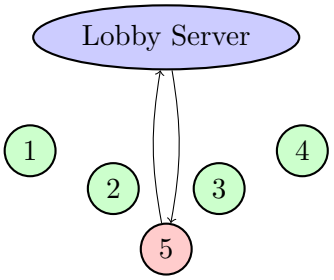
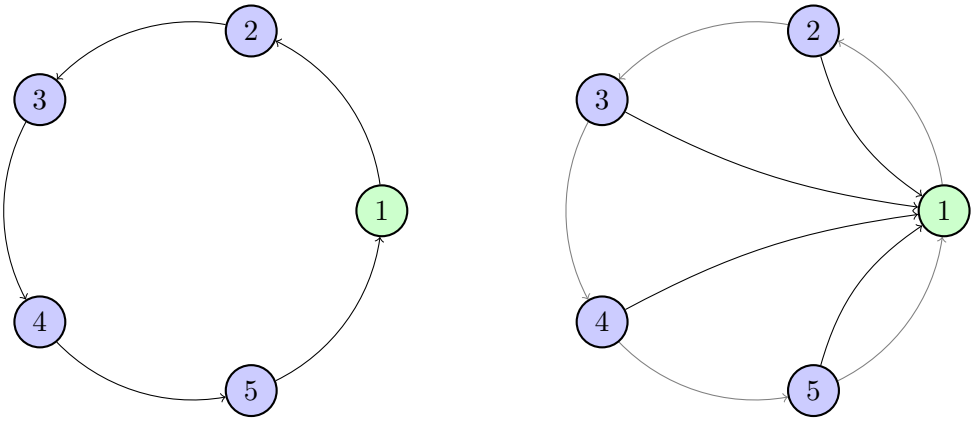


Figura 4: Eccezione del server alla richiesta di una lobby chiusa

3.4.2 Lo schema con leader dinamico.

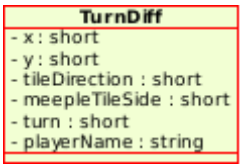
Lo schema di distribuzione si basa sull'elezione di un leader "dinamico", questo leader e' deciso in base ad un lancio di un dado iniziale (nella realta' implementato come un random intero a 32 bit, per evitare lanci ripetuti), questo lancio viene quindi distribuito dai vari player ad ogni giocatore, creando un **ordine** di gioco, il giocatore con punteggio maggiore verra' quindi dichiarato come leader corrente, e da li a decrescere.
La topologia risulta quindi una rete monodirezionale con passaggio del testimone (token ring) per la decisione del leader corrente.



3.4.3 Aggiornamenti allo stato

Uno degli aspetti piu' importanti del sistema di comunicazione del gioco e' la presenza di classi di differenza (classe **TurnDiff**), le suddette classi rendono la comunicazione il piu' leggero possibile essendo composte da:

- Le coordinate della tessera;
- la rotazione della tessera posizionata;
- le posizioni relative del meeple se presente;
- il turno e il giocatore correnti.



3.4.4 Tolleranza ai guasti

Il sistema risulta tollerante ai guasti di tipo crash. Nel caso di un guasto di tipo crash sul nodo leader corrente, sara' il sistema ad invocare un'eccezione di tipo **RemoteException** verso i nodi interroganti, che lo elimineranno quindi dalla loro lista di interrogazione, riconfigurando l'anello.
Nel caso di un guasto di un nodo intermedio il risultato sara' il medesimo al momento di un'interrogazione da parte dei vari nodi dell'anello.
Questo genere di configurazione mantiene coerente lo stato locale delle diverse istanze, poiche' ogni nodo aspetta la risoluzione delle varie mosse ad esso precedenti prima di

essere interrogato a sua volta e poter agire.
Questo schema e' banalmente possibile utilizzando una semplice rete di tipo token ring, ma e' stato scelto di implementare il tutto come una sorta di cricca per l'aggiornamento automatico e la visualizzazione dei risultati con latenze brevi: se si fosse ponderato per una struttura completamente circolare (utilizzante lo stesso modello logico) gli aggiornamenti allo stato locale sarebbero applicati solamente dopo che il controllo (e quindi la leadership) fosse tornata al nodo richiedente, risultando in un'attesa pari ad **N-1** turni; essendo il gioco in questione un gioco di logica non propriamente reattivo e con turni di gioco potenzialmente molto lunghi e riflessivi e' stato optato per un modello piu' pesante da un punto di vista di scambio di informazioni che da un punto di vista piu' leggero come comunicazioni ma, allo stesso tempo, meno reattivo per tutti i client.

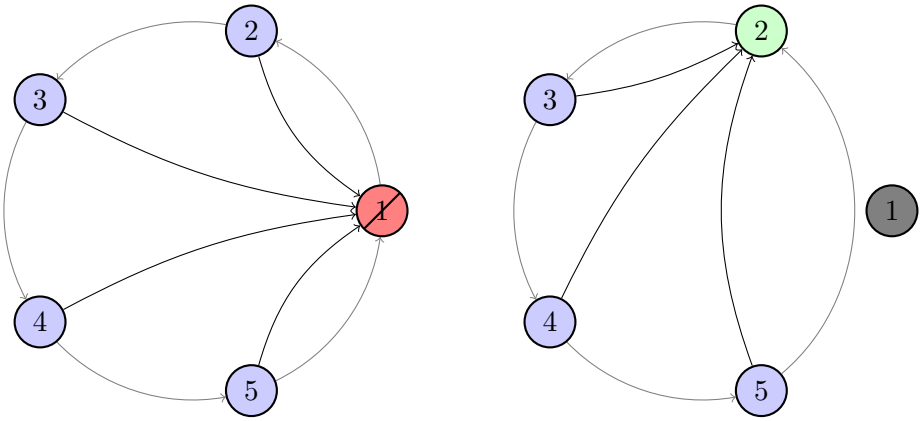


Figura 5: Riconfigurazione dei client dovuta ad un crash

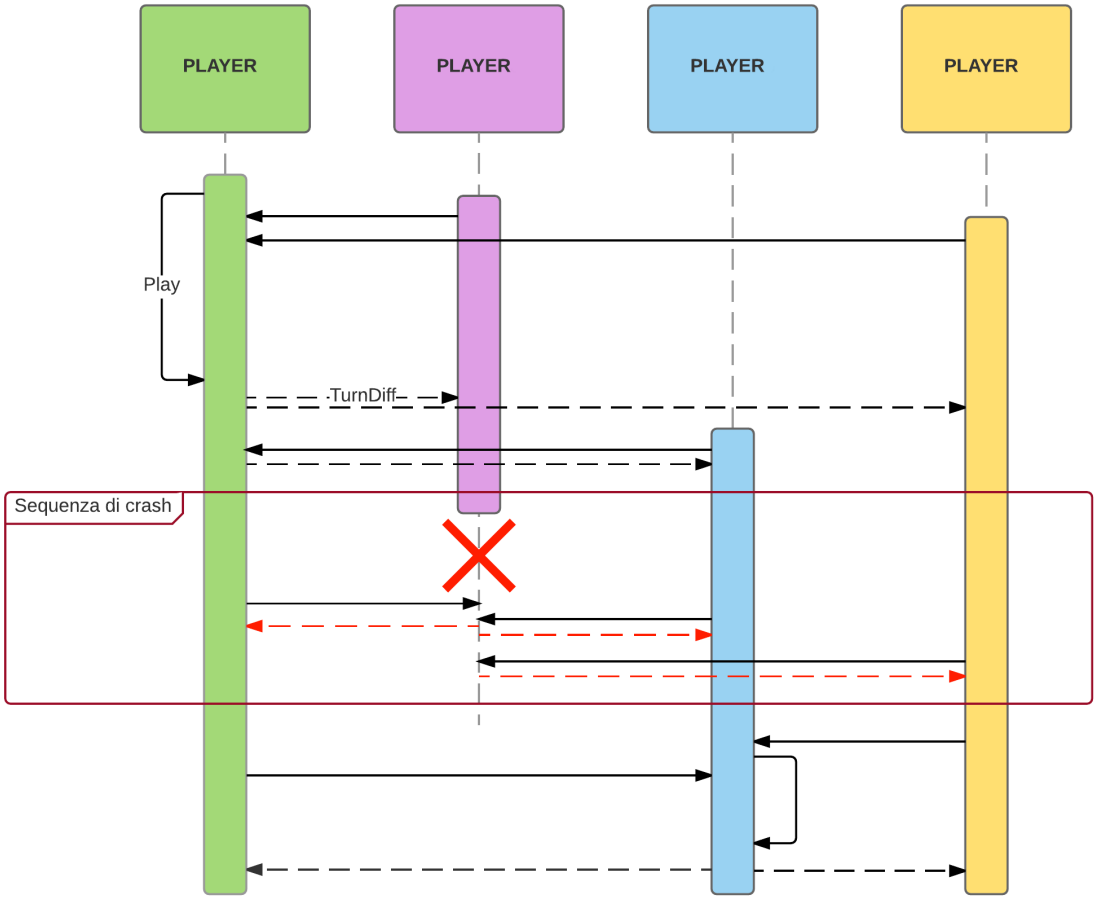


Figura 6: Comunicazione in una partita d'esempio a 4 giocatori

4 Valutazione

5 Conclusioni