

Raga.io

Grow, eat and survive in a distributed chaos.

Indice

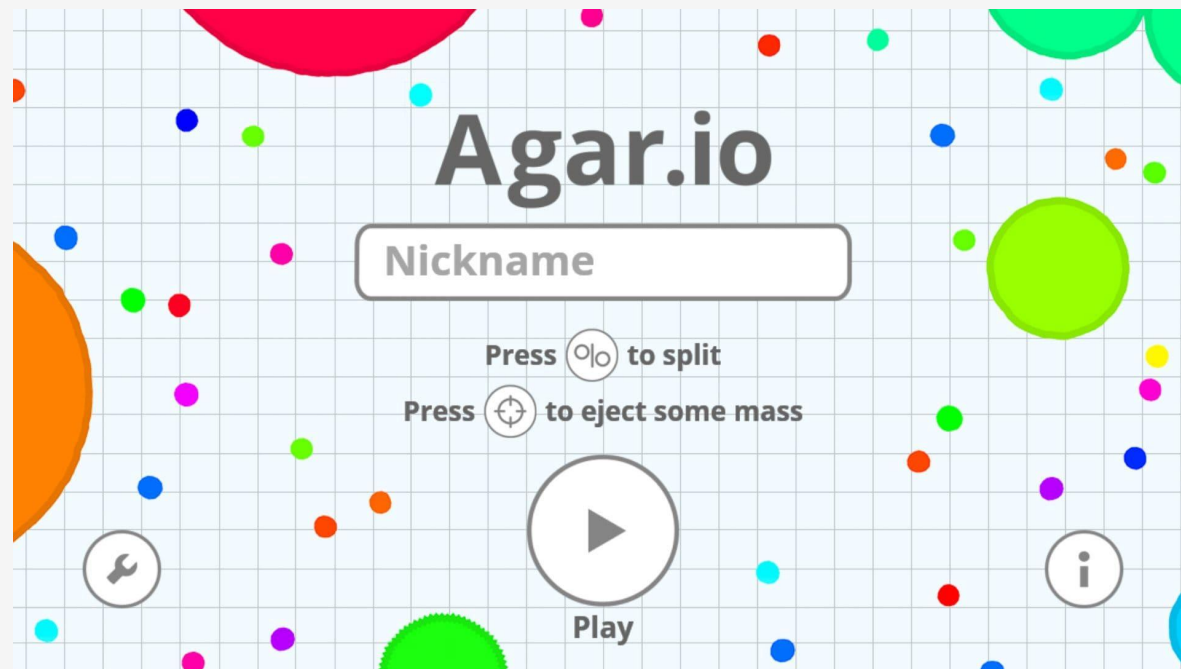
- I. Introduzione
 - A. Agar.io
- II. Requisiti
 - A. Requisiti funzionali
- III. Design
 - A. Architettura
 - B. Infrastruttura
 - C. Modellazione
 - D. Interazioni
 - E. Comportamento
 - F. Data consistency
 - G. Fault tolerance
 - H. Sicurezza
- IV. Conclusioni
- V. Demo



Introduzione

Agar.io

È un gioco d'azione **multiplayer** di massa nel quale ogni giocatore controlla una **cellula** che può mangiare altre cellule - più piccole - in giro per la mappa. L'obiettivo per un giocatore è mangiare tutte le altre cellule per **crescere** ed **evitare di essere mangiato**.



Requisiti

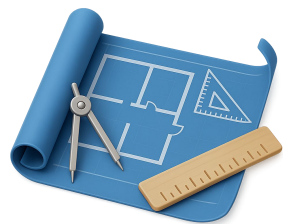
Requisiti Funzionali

- **Gameplay:** I giocatori controllano cellule colorate che crescono mangiando cibo o avversari più piccoli (rallentando all'aumentare della stazza); il sistema gestisce le collisioni in base alla dimensione.
- **Gestione sessioni e UI:** L'interfaccia grafica deve permettere di creare nuove partite, unirsi a sessioni casuali o specifiche tramite ID univoco e gestire nickname e statistiche.
- **Multiplayer:** Il sistema deve garantire interazione in tempo reale a bassa latenza e supportare la concorrenza di più giocatori.
- **Architettura Scalabile:** È richiesto un backend con Master Server per coordinare server di gioco multipli, gestendo allocazione dinamica, scalabilità orizzontale e procedure di backup/failover.

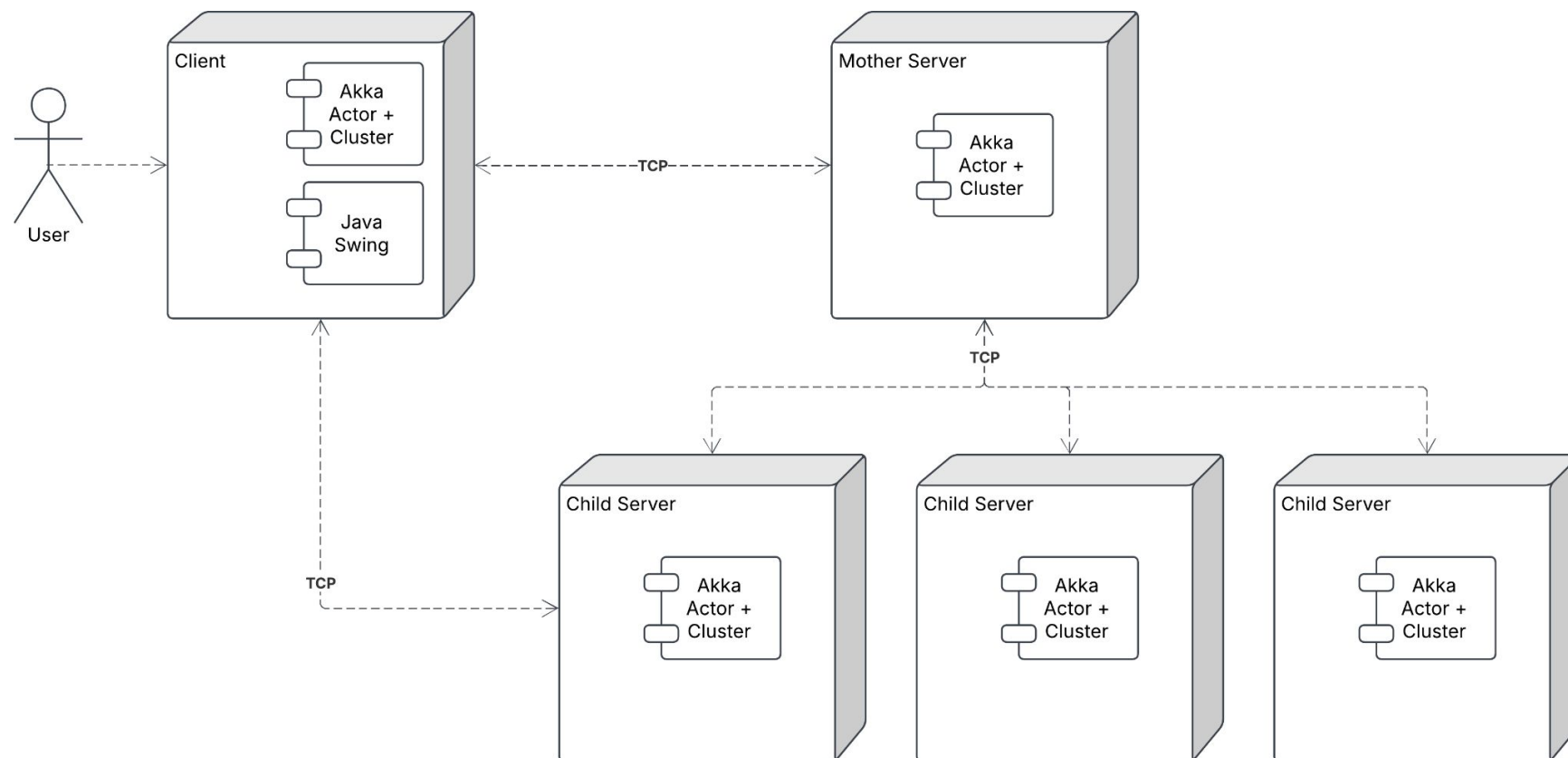
```
1 // Acceptance Criteria
2 def validateSystem():
3     assert fps >= 30 &&
4     assert playerSupported >= 5 &&
5     assert visibleCellGrowing
```

Design

Architettura



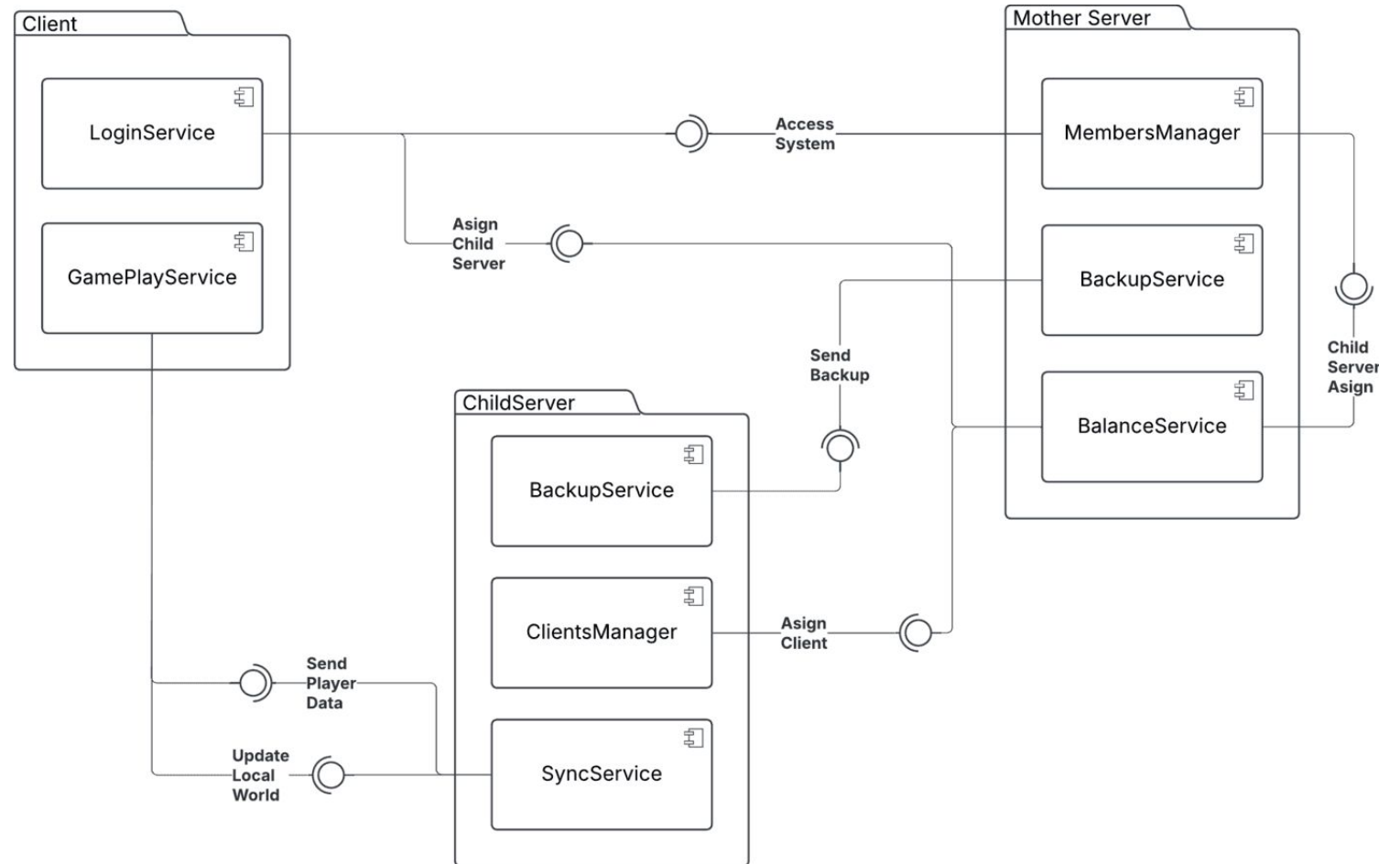
Hierarchical Distributed Architecture



Infrastruttura

Il sistema si sviluppa su 3 tipi di nodi che comunicano secondo il paradigma di Akka Actors su un Akka Cluster:

- Client: Applicazione interagibile mediante GUI Java Swing.
- Mother Server: Il coordinatore centrale del cluster.
- Child Server: Nodo worker che gestisce una specifica sessione di gioco.



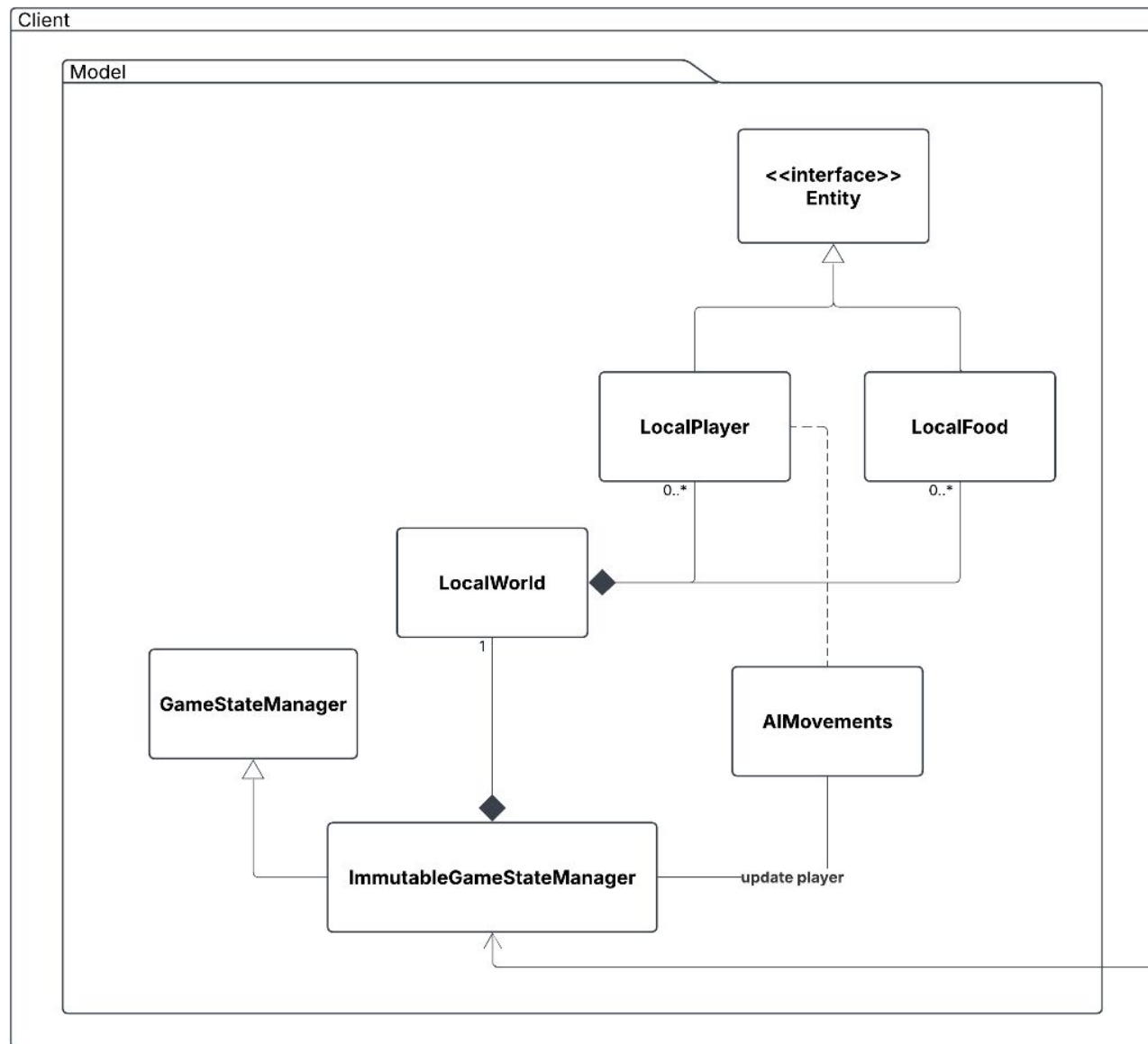
Modellazione



Client - MVC

Model

- **Entity (interfaccia)**: astrazione comune implementata da LocalPlayer e LocalFood.
- **LocalPlayer / LocalFood**: entità presenti nel mondo di gioco.
- **LocalWorld**: rappresenta l'ambiente di gioco, contenente giocatori e oggetti cibo.
- **ImmutableGameStateManager**: gestiscono lo stato del gioco; la versione immutabile offre sola lettura.
- **AIMovements**: aggiorna i movimenti e il comportamento dei giocatori AI.



Modellazione



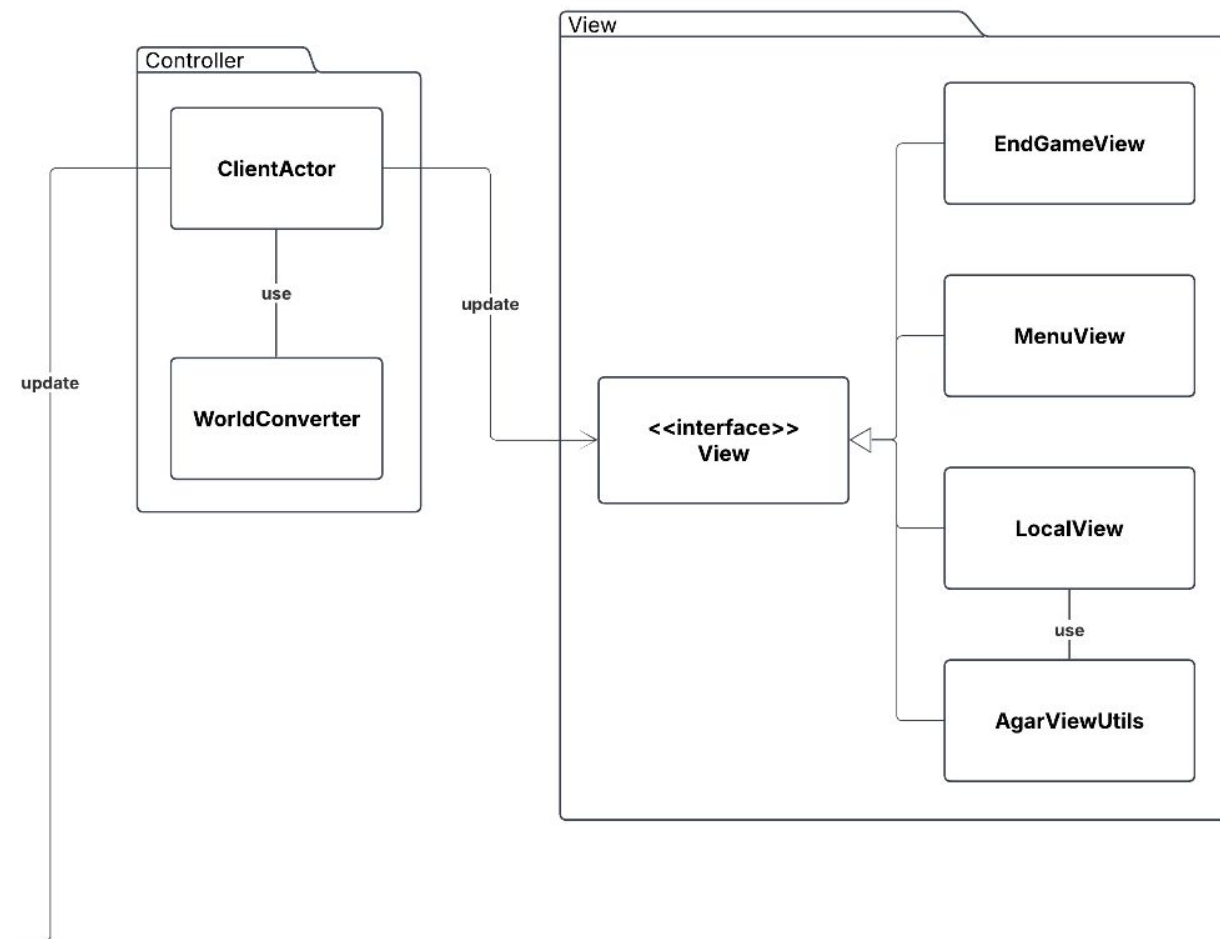
Client - MVC

Controller

- **ClientActor**: controller centrale che gestisce il flusso di gioco e la comunicazione con i server.
- **WorldConverter**: utilizzato da ClientActor per convertire i dati del modello in un formato adatto alle elaborazioni locali.

View

- **View (interfaccia)**: astrazione generale per i componenti visivi.
- **MenuView / EndGameView / LocalView**: diverse rappresentazioni grafiche: menu, gameplay, schermata finale.
- **AgarViewUtils**: funzioni di utilità per disegnare gli elementi del gioco nella LocalView.

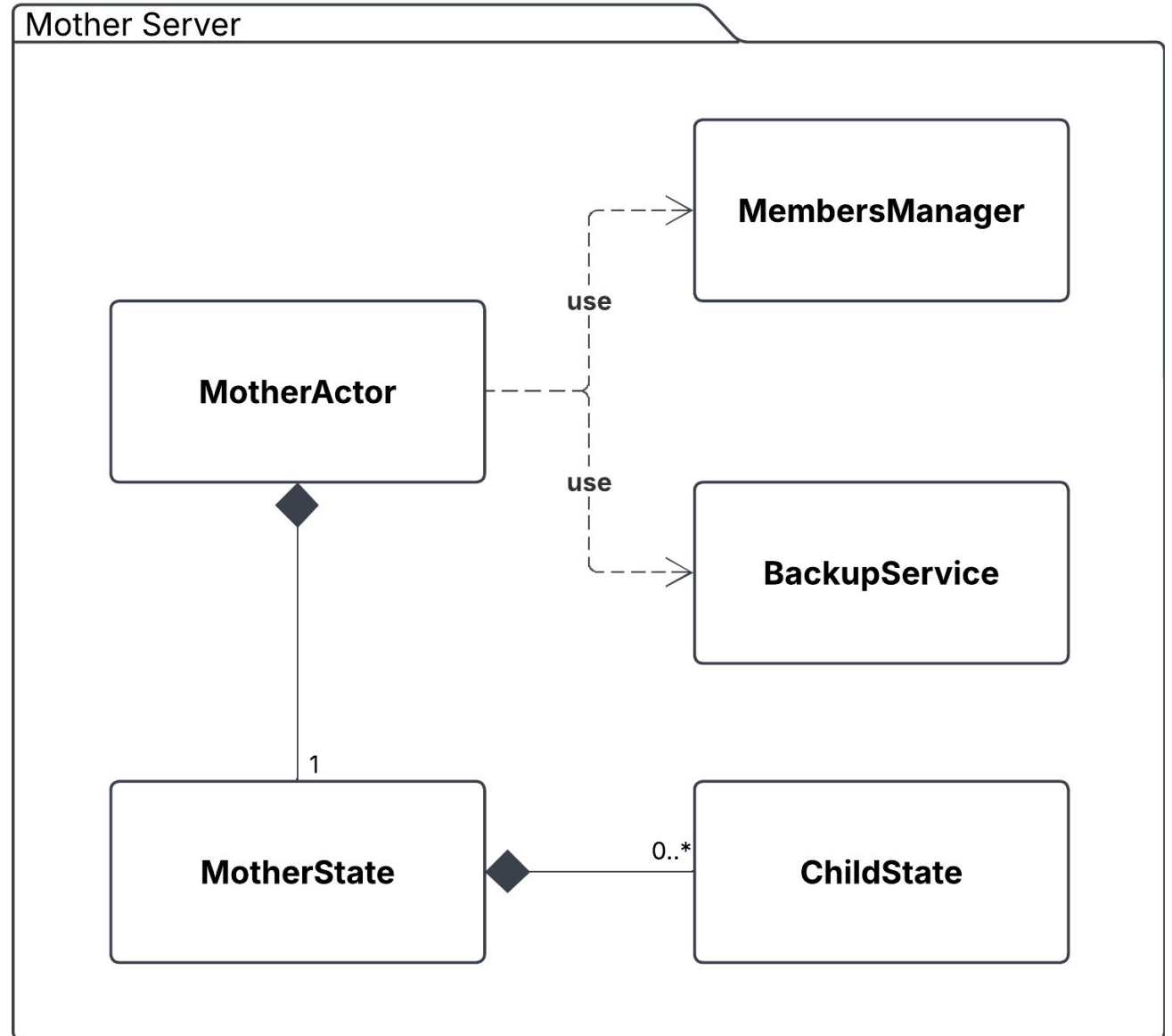


Modellazione



Mother Server

- **MotherActor**: attore principale del sistema.
- **Members Manager**: gestisce la scoperta di Client e Child Server che si connettono al sistema.
- **BackupService**: si occupa delle operazioni di backup dei dati.
- **MotherState**: contiene i riferimenti dei ChildServer, dei Client in attesa ed i ChildState.
- **ChildState**: descrive lo stato di un Child Server, includendo il carico e l'ID del mondo gestito.

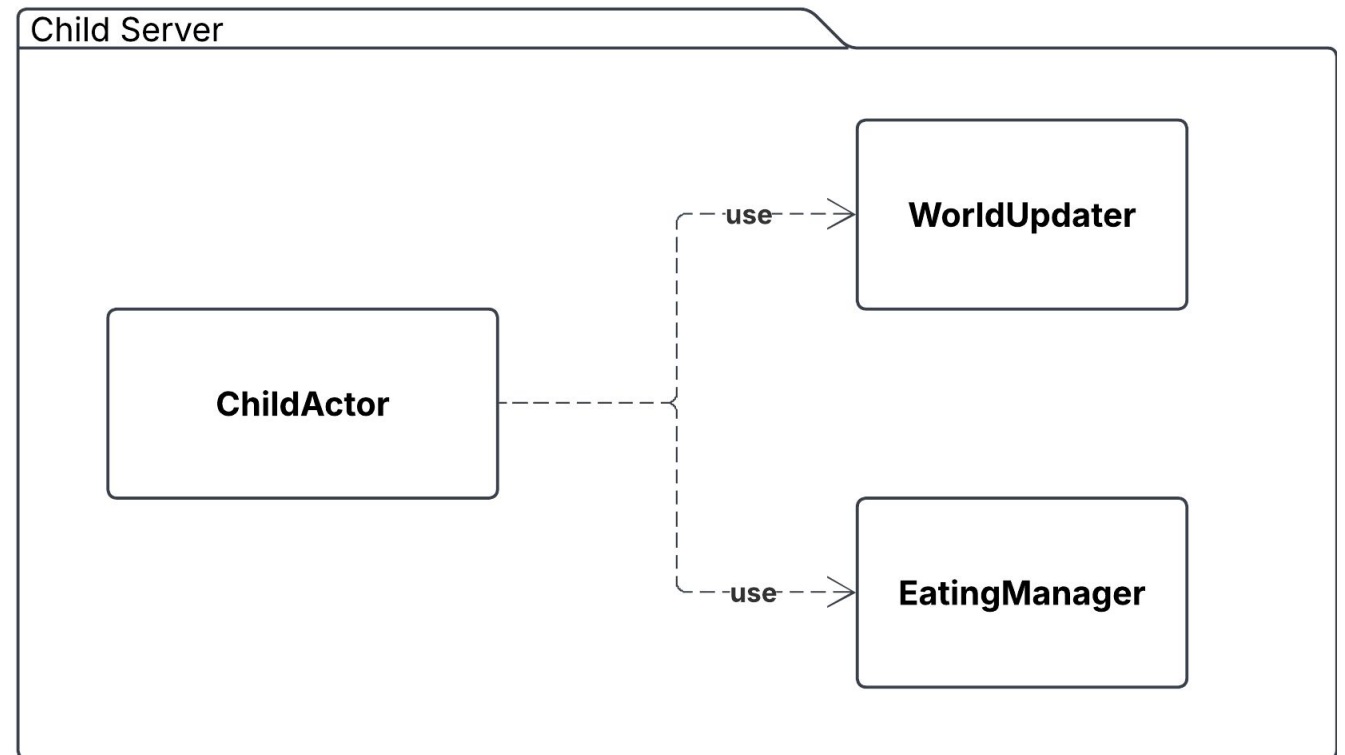


Modellazione



Child Server

- **Child Actor**: riceve gli input dei giocatori, aggiorna lo stato del mondo e lo invia a tutti i partecipanti.
- **World Updater**: calcola le trasformazioni del mondo in base agli input dei giocatori.
- **Eating Manager**: gestisce le collisioni e determina quando un giocatore può mangiare altri giocatori o il cibo.

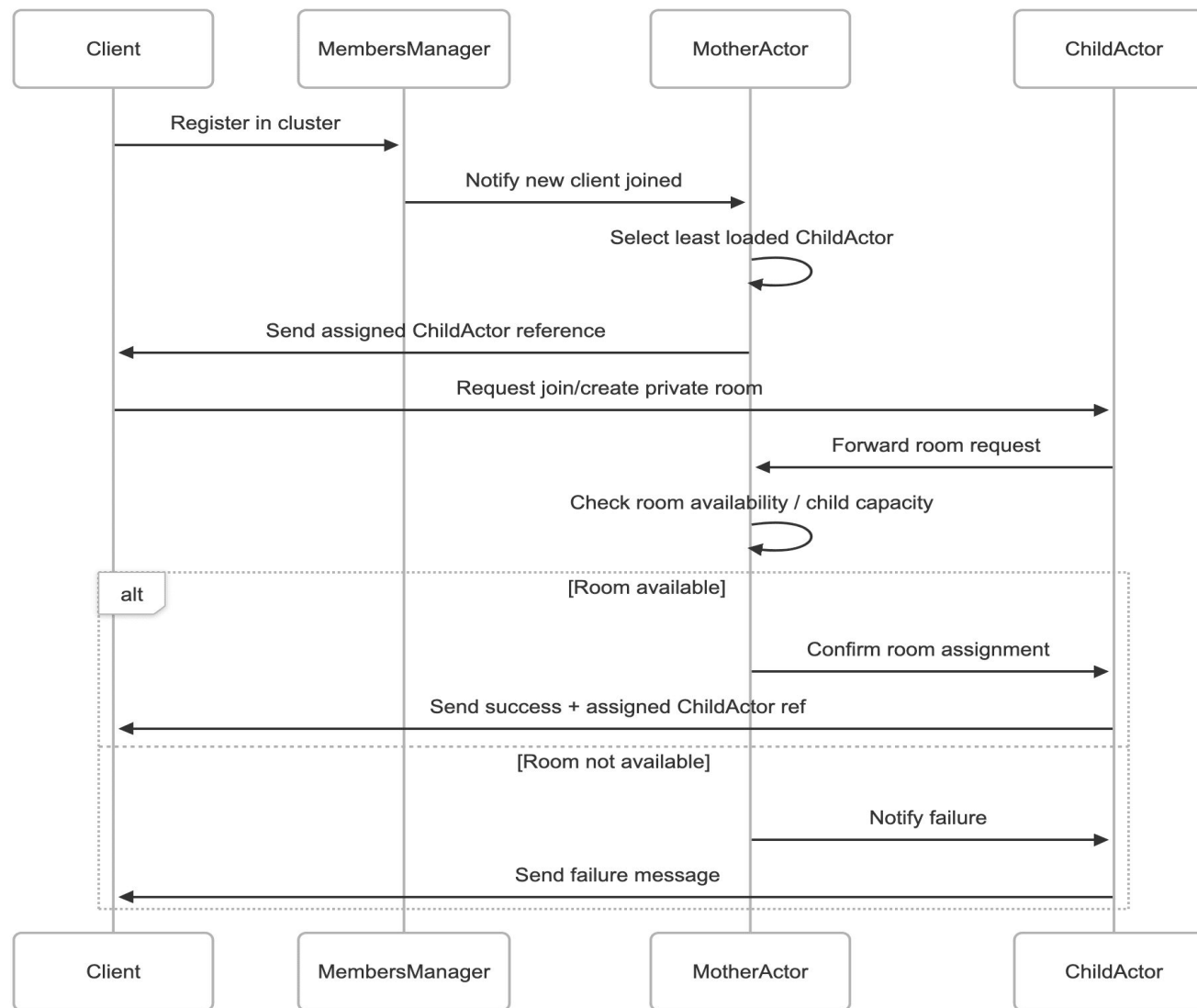


Interazioni

Client - MotherServer

Il client viene assegnato ad un ChildActor scarico scelto dal MotherServer, il ChildActor selezionato avrà il compito di gestire le richieste del Client:

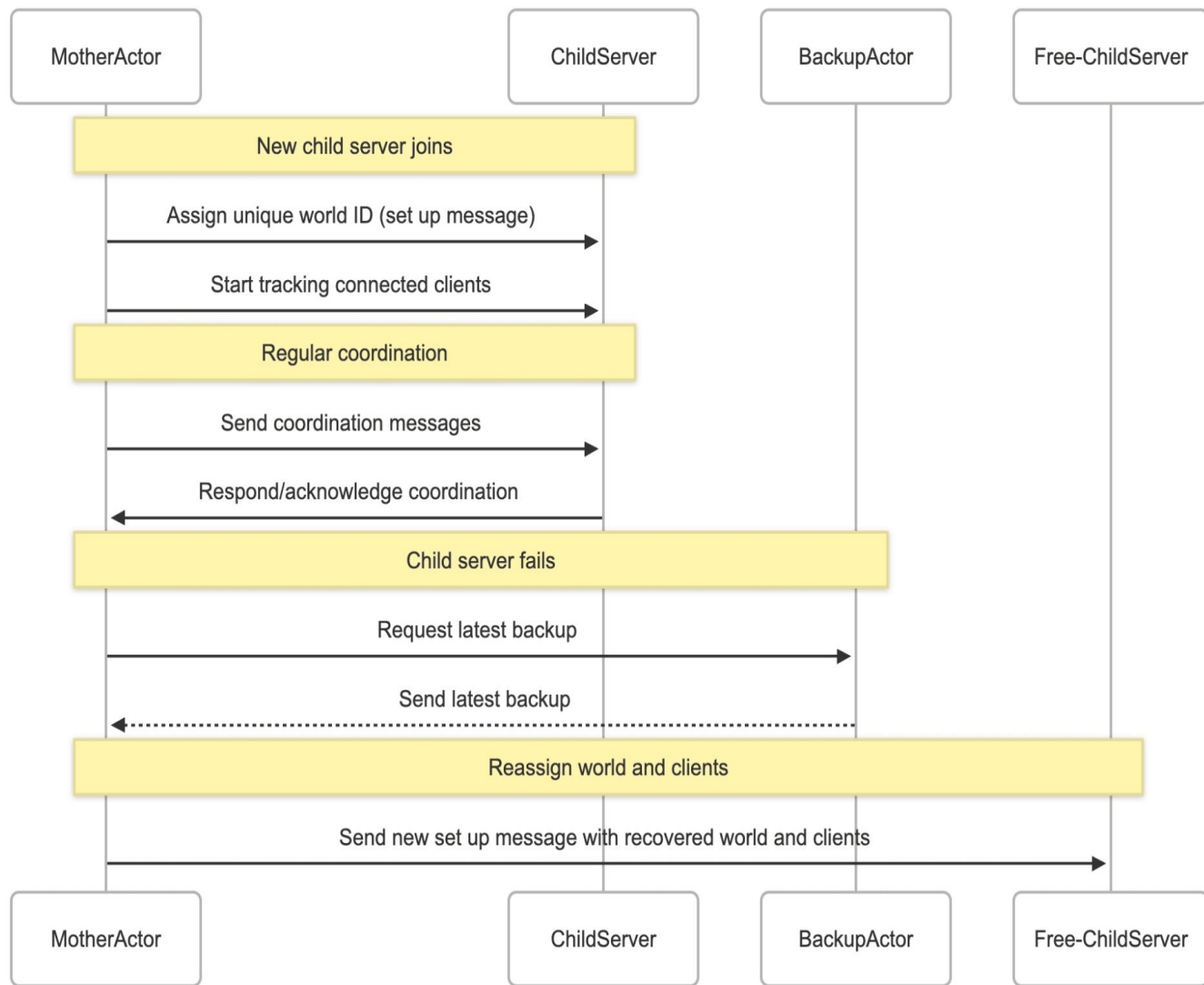
- entrare in una stanza
 - random
 - privata
- creazione di una stanza privata



Interazioni

Mother - ChildServer

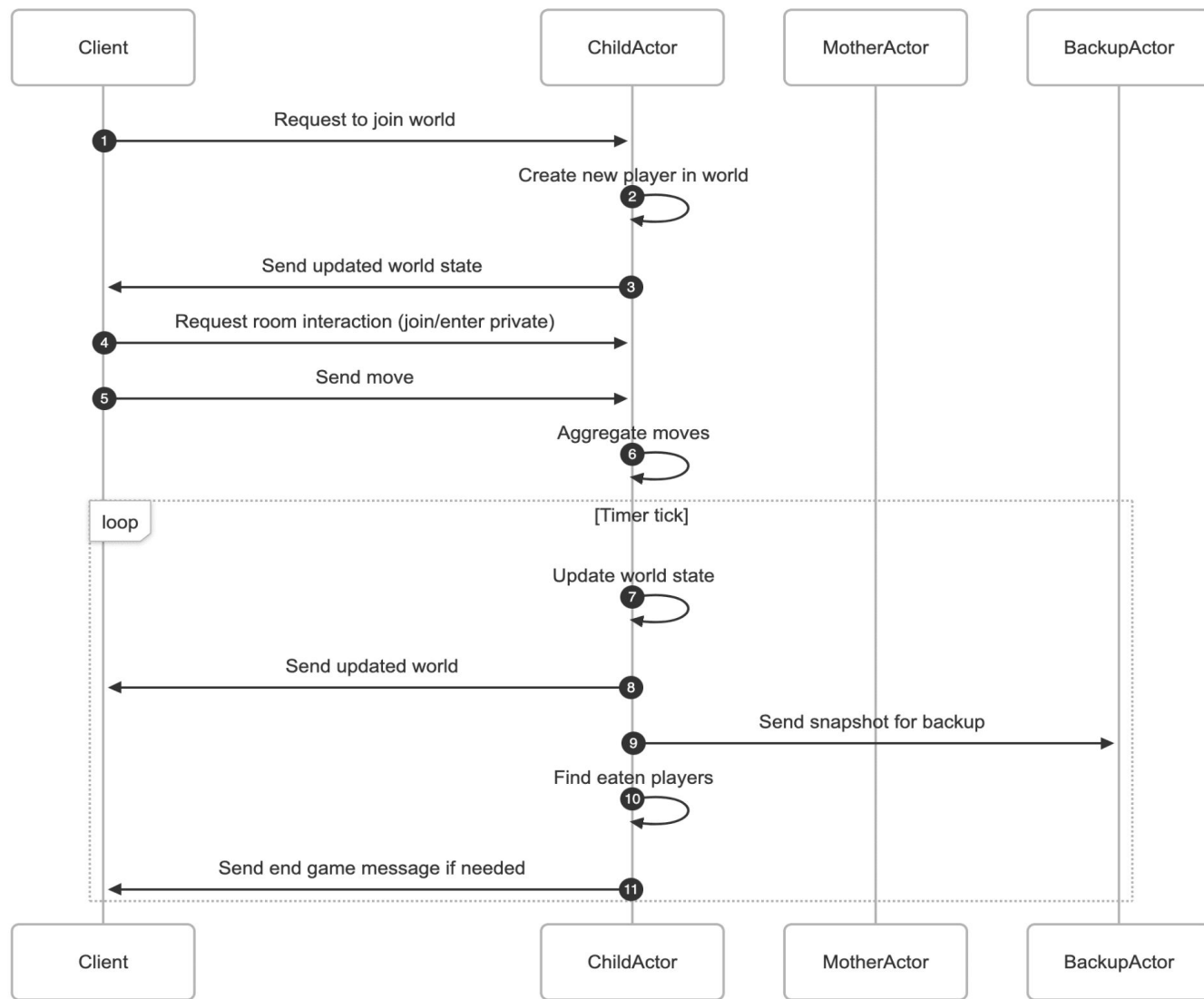
Il MotherActor garantisce la continuità del servizio monitorando i ChildServer e, in caso di guasto di quest'ultimi, recupera i dati dal BackupActor per trasferire immediatamente tutto il carico di lavoro su un ChildServer libero.



Interazioni

ChildServer - Client

Il Client si unisce ad una partita ed il ChildActor responsabile gli assegna una posizione libera. Il ChildActor, con una certa frequenza, aggiorna lo stato del mondo in base alle mosse ricevute e invia il nuovo mondo a tutti i Client connessi ed al BackupActor, inoltre verifica le condizioni per l'eventuale termine della partita dei Client.



Comportamento

- **Architettura Message-Driven:** Sistema distribuito basato esclusivamente sullo scambio di messaggi.
- Separazione dei ruoli:
 - **Stateful** (es. MotherActor): Centralizzano i dati critici e lo stato di gioco.
 - **Stateless** (es. MembersManager): Gestiscono discovery ed eventi senza persistenza.
- **Scalabilità:** Gestione ottimizzata di alti volumi di eventi asincroni.
- **Reattività:** Transizioni di stato attivate solo da messaggi o eventi del cluster.

Data consistency

- **Problemi di Serializzazione:** Nodi con versioni differenti del protocollo di comunicazione possono causare errori di deserializzazione o perdita di messaggi.
- **Stato Non Deterministico:** L'ordine di elaborazione delle mosse non è garantito e ciò determina lo stato del mondo inviato dal Child Server.
- **Elaborazione Ritardata:** La consegna dei messaggi rischia di essere computata in tick differenti del ChildServer.
- **Volatilità dei Dati:** Non esiste persistenza su disco; lo stato risiede interamente in memoria e viene perso al termine della sessione.

Fault tolerance



Failure Detection

Il Mother Server è costantemente aggiornato sui nodi che entrano a far parte del Cluster e su quelli che per un qualche motivo si disconnettono.

Akka - under the hood - utilizza l'heart-beating per il monitoraggio.



Snapshot & Backup

I Child Server inviano periodicamente lo stato del mondo al BackupService (attore generato dal Mother Server). In caso di crash, l'ultimo stato valido viene ripristinato.

Security



Il Mother Server è isolato e nascosto dietro i Child Server.

La compromissione di un nodo periferico non intacca il sistema centrale.

Conclusioni

Mother Server: il coordinatore e SPF



Matchmaking

Gestisce l'ingresso dei nuovi client e li indirizza verso i Server di gioco disponibili.



Load-balancing

Distribuisce il carico selezionando il Child Server meno congestionato per i nuovi giocatori.



Recovery

Supervisiona lo stato dei Child Server. In caso di crash, ripristina la sessione su un nodo funzionante.

Child Server: engine di gioco



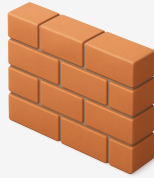
Server Authoritative

Il Child Server è l'unica fonte di verità (Single Source of Truth). Riceve gli input dai client, calcola la fisica (movimento, collisioni, crescita) e invia lo stato aggiornato del mondo.



Game Loop Sincrono

Aggiorna lo stato del mondo a intervalli regolari (tick). Questo approccio garantisce che tutti i client vedano una rappresentazione consistente della mappa, prevenendo il cheating.



Limitazioni

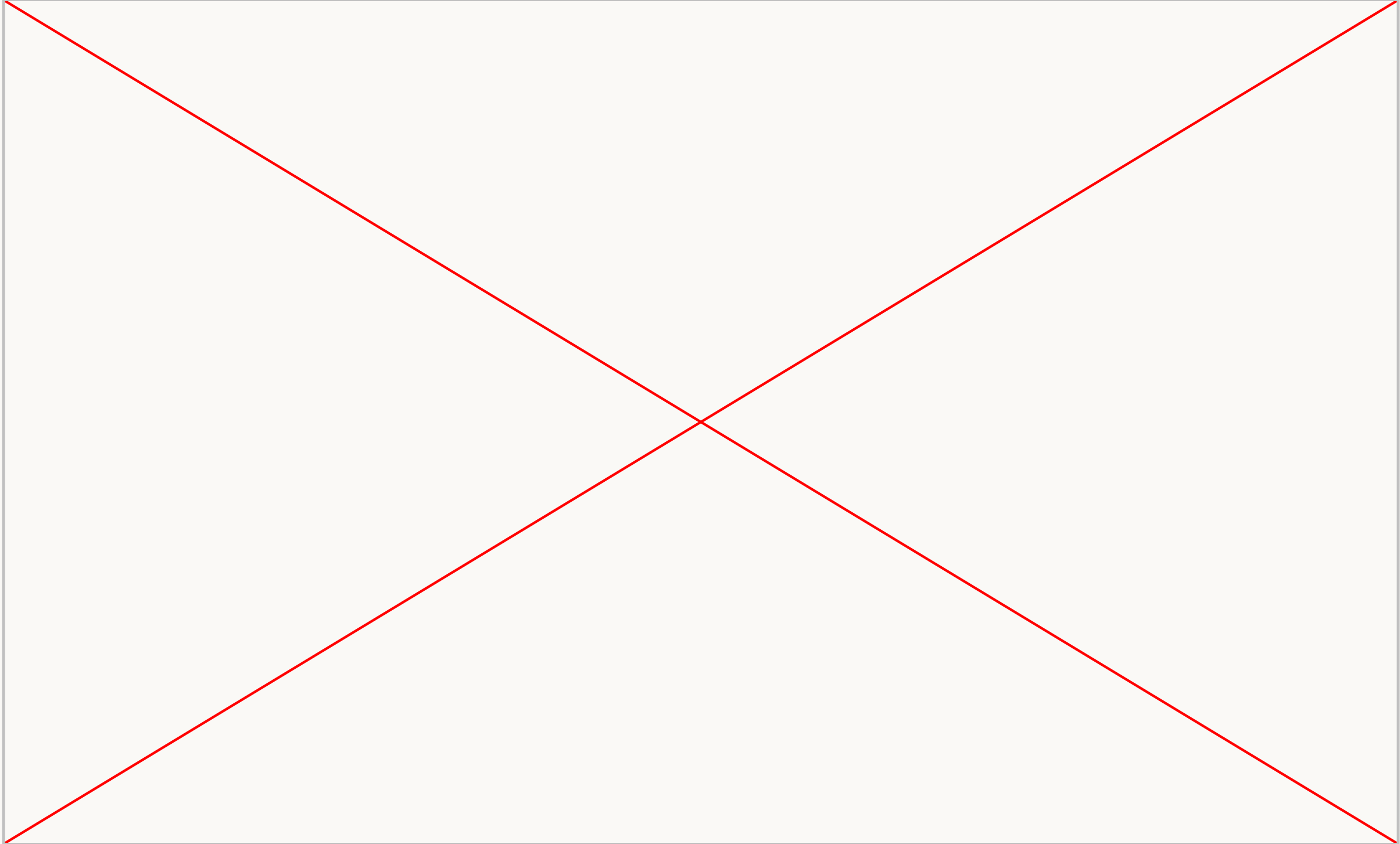
- Il Mother Server è il **Single Point of Failure**
- Il recupero a snapshot può causare **perdita di azioni recenti**.
- La distribuzione del carico si basa solo sul numero di giocatori, **non si considerano latenza o capacità dei server**.
- In condizioni di stress della rete l'**availability può venire meno**.

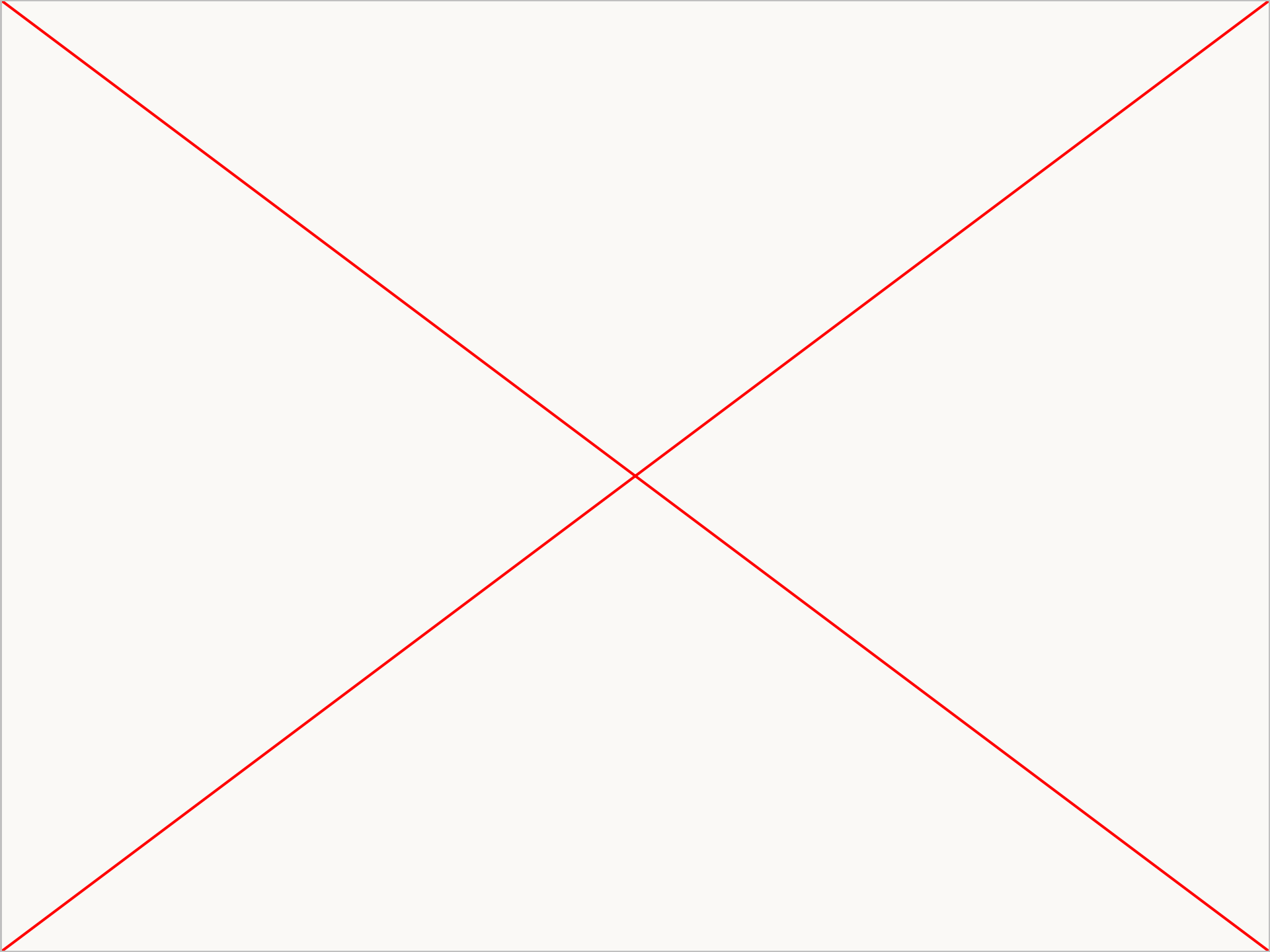


Sviluppi futuri

- **Transizione ad UDP** per gli update di movimento in tempo reale e TCP solo per eventi critici, riducendo la latenza.
- Il client **anticipa i movimenti e li riconcilia** con lo stato fornito dal server, migliorando fluidità e reattività.
- **Salvare statistiche, classifiche e personalizzazioni** tramite un **database** per mantenerle tra diverse sessioni di gioco.

Demo





Grazie per l'attenzione.

`github.com/elvisperluka/raga.io`