Distributed Computing

agar.io game

Autore: Sajmir Buzi

Anno Accademico 2025

Contents

1 Introduzione

Il progetto nasce con l'obiettivo di realizzare una versione distribuita e concorrente del celebre gioco Agar.io, in cui più giocatori e intelligenze artificiali si muovono e competono in un ambiente condiviso. La simulazione deve essere in grado di gestire molteplici entità che interagiscono in tempo reale, garantendo sia la correttezza dello stato globale sia una buona reattività dell'interfaccia utente. Per raggiungere questi obiettivi, il progetto adotta un'architettura basata su attori (Actor Model) e una comunicazione esclusivamente tramite messaggi, sfruttando le potenzialità offerte dal framework Akka.

2 Analisi del Problema

Il progetto consiste nello sviluppo di una versione distribuita e concorrente del gioco *Agar.io*, dove più giocatori (umani e AI) si muovono in un mondo condiviso, mangiano cibo e si sfidano tra loro.

La sfida principale riguarda la gestione sicura e reattiva della concorrenza tra molteplici attori che interagiscono in tempo reale: giocatori, AI, cibo e il gestore del mondo.

Aspetti di concorrenza rilevanti

- Ogni giocatore e ogni AI agisce in modo indipendente, inviando comandi e ricevendo aggiornamenti dal mondo.
- Il mondo deve gestire simultaneamente molteplici richieste di movimento, collisione e aggiornamento dello stato.
- La generazione e la rimozione del cibo avvengono in parallelo rispetto alle azioni dei giocatori.
- L'interfaccia grafica locale deve essere aggiornata in tempo reale, senza bloccare la simulazione.
- È necessario evitare condizioni di gara e garantire la coerenza dello stato globale, anche in presenza di latenze di rete e di attori distribuiti.

3 Descrizione del Design, Strategia e Architettura Adottata

3.1 Architettura Generale

Il sistema è progettato secondo il modello **Actor** di Akka Typed, che permette di gestire la concorrenza e la distribuzione tramite scambio di messaggi tra attori.

Componenti principali:

- `WorldManagerActor`: attore centrale che mantiene lo stato globale del mondo, gestisce i movimenti, le collisioni, la generazione del cibo e la registrazione dei giocatori.
- `PlayerActor`: rappresenta ogni giocatore (umano o AI) come attore indipendente. Invia comandi di movimento, riceve aggiornamenti sullo stato del mondo e gestisce la logica di crescita e morte.
- `FoodManagerActor`: gestisce la generazione, la distribuzione e la rimozione del cibo nel mondo.
- `DistributedLocalView`: interfaccia grafica locale che mostra lo stato del mondo al giocatore e permette l'interazione in tempo reale.
- **Protocollo di messaggi**: tutte le interazioni tra attori avvengono tramite messaggi tipizzati, che garantiscono sicurezza e chiarezza nella comunicazione.

1. `actors`

Questa cartella contiene tutti gli attori Akka utilizzati per gestire la logica concorrente e distribuita del gioco. Ogni attore ha un compito ben preciso:

- `WorldManagerActor`: è l'attore principale che mantiene lo stato globale del mondo. Si occupa di aggiornare i movimenti, gestire le collisioni tra entità, generare il cibo e registrare nuovi giocatori.

- `PlayerActor`: rappresenta ogni singolo giocatore (sia umano sia AI). Gestisce i comandi di movimento, l'aumento di massa quando mangia e le eventuali condizioni di morte. Comunica inoltre con la vista locale del giocatore.

- `FoodManagerActor`: è l'attore incaricato di creare, distribuire e rimuovere il cibo nello spazio di gioco.

- `PlayerLogicActors`: contiene la logica specifica di ogni giocatore, come il comportamento dell'intelligenza artificiale e le interazioni personalizzate con il mondo.

Questa organizzazione consente di separare chiaramente le responsabilità e di sfruttare appieno i vantaggi della concorrenza e della distribuzione offerti dal modello Actor di Akka.

2. `controller`

In questa cartella è presente la logica di avvio e coordinamento generale dell'applicazione. Il cuore è rappresentato dal file:

- `Main`: funge da punto di ingresso dell'applicazione. Qui vengono creati gli attori principali, avviata la simulazione e impostata la configurazione iniziale del sistema.

Eventuali controller aggiuntivi possono essere introdotti per coordinare aspetti più specifici dell'orchestrazione tra attori e componenti distribuiti.

3. `distributed`

Questa sezione contiene il codice dedicato alla gestione della distribuzione del sistema su più nodi o processi. È qui che avviene la vera comunicazione tra server e client remoti.

- `GameServer` e `GameClient`: gestiscono la connessione e lo scambio di informazioni tra più giocatori distribuiti su macchine diverse.

- `AIClient`: gestisce le istanze dei giocatori controllati dall'intelligenza artificiale, eseguendole anche su nodi remoti.

In questa cartella si trovano anche le logiche legate alla sincronizzazione tra nodi e all'eventuale configurazione del cluster.

4. `model`

Questa cartella contiene tutte le classi che definiscono lo stato del gioco, ossia i modelli dati che descrivono entità come giocatori, cibo e mondo.

- `Player`: rappresenta le informazioni associate a un giocatore, come la posizione, la massa e il punteggio.
- `Food`: rappresenta una singola unità di cibo nel mondo di gioco.
- `World`: tiene traccia dello stato globale del mondo, inclusi tutti i giocatori e il cibo presenti.

Sono inoltre presenti classi di supporto per gestire logiche come il movimento, le collisioni e la crescita. Questa separazione tra logica degli attori e modello dati migliora l'organizzazione del progetto.

5. `protocol`

In questa cartella sono definiti tutti i messaggi e i protocolli utilizzati per la comunicazione tra attori. Ogni messaggio è ben tipizzato, rendendo la comunicazione sicura e chiara.

- `PlayerActorMessage`, `WorldManagerMessage`, `FoodManagerMessage`: rappresentano i diversi tipi di messaggi che gli attori si scambiano per coordinare il comportamento del sistema.

Questa scelta progettuale facilita la manutenzione e riduce al minimo gli errori legati alla concorrenza, poiché ogni interazione tra attori è esplicitamente dichiarata.

6. `view`

Infine, questa cartella gestisce tutto ciò che riguarda l'interfaccia grafica e l'interazione dell'utente con il sistema.

- `DistributedLocalView`: è la GUI che mostra al giocatore lo stato aggiornato del mondo e consente l'interazione in tempo reale.
- `AgarViewUtils`: contiene funzioni di utilità per il rendering grafico e altre operazioni grafiche comuni.

Sono inoltre presenti eventuali pannelli e componenti Swing per visualizzare e controllare il gioco. Tutti gli aggiornamenti alla GUI vengono eseguiti in modo *thread-safe*, così da garantire una buona esperienza utente senza blocchi o rallentamenti.

3.2 Strategia Concorrente

- **Isolamento degli attori**: ogni attore incapsula il proprio stato e comunica solo tramite messaggi, evitando la condivisione diretta di dati e quindi le condizioni di gara.
- **Aggiornamenti periodici**: il mondo e il cibo vengono aggiornati tramite timer e messaggi periodici, garantendo la reattività della simulazione.
- **Gestione delle collisioni**: il `WorldManagerActor` si occupa di verificare le collisioni tra giocatori e cibo, aggiornando lo stato globale in modo atomico.
- **UI thread-safe**: tutte le operazioni sulla GUI sono eseguite nel thread Swing, garantendo la sicurezza e la reattività dell'interfaccia.
- **Scalabilità**: la struttura ad attori permette di gestire facilmente molti giocatori e AI, sfruttando il parallelismo offerto da Akka.

3.3 Punti di Forza del Design

- **Separazione delle responsabilità**: ogni componente ha un ruolo chiaro e comunica solo tramite interfacce ben definite.
- **Reattività**: l'utente può interagire con la simulazione in qualsiasi momento, senza blocchi o rallentamenti.
- **Sicurezza concorrente**: l'uso degli attori e dei messaggi tipizzati elimina la possibilità di accessi concorrenti non controllati.
- **Manutenibilità ed estendibilità**: la struttura modulare facilita l'estensione e la modifica del sistema.

3.4 Conclusioni

Il progetto dedicato ad **Agar.io** rappresenta un valido esempio di come sia possibile realizzare un sistema moderno e distribuito, capace di gestire la concorrenza in modo efficace. La chiara separazione tra logica applicativa, gestione dei dati, protocolli di comunicazione e interfaccia grafica ha reso il sistema robusto, scalabile e semplice da sviluppare e mantenere.

Grazie all'utilizzo del modello *Actor* offerto da Akka, la gestione della concorrenza e della distribuzione risulta naturale e sicura. Questo approccio consente di mantenere l'interfaccia utente sempre fluida e reattiva, pur in presenza di molteplici entità che interagiscono tra loro in tempo reale. Inoltre, la modularità del sistema ne facilita l'estensione futura con possibili aggiunte che non influenzano la reattività del gioco.