PCD Assignment 03 - Agar.io with Java RMI

A cura di

Alessandra Versari - alessandra.versari2@studio.unibo.it

Lorenzo Rigoni - lorenzo.rigoni2@studio.unibo.it

Riccardo Moretti - riccardo.moretti6@studio.unibo.it

Analisi del problema

L'obiettivo di questo assignment è trasformare l'implementazione centralizzata del gioco Agar.io (creata da G. Aguzzi) in una versione distribuita con server centralizzato che permetta a più giocatori di connettersi da macchine diverse, mantenendo le meccaniche di gioco originali. Questa versione del gioco sarà implementata tramite *Java RMI*.

Le principali meccaniche di cui tenere conto sono:

- **Movimento**: Il blob si muove seguendo il cursore del mouse.
- **Mangiare altri player**: Se un giocatore tocca un blob più grande, viene mangiato e eliminato.
- **Crescita**: Mangiando palline di cibo (verdi) o altri giocatori più piccoli, la massa del blob aumenta.

Design, strategia ed architettura

Per affrontare il problema dell'implementazione di una versione distribuita, abbiamo adottato un'architettura client-server basata su **Java RMI (Remote Method Invocation)**. Questo approccio consente una separazione chiara tra il lato server (che gestisce la logica di gioco e lo stato globale) e il lato client (che si occupa dell'interfaccia grafica e dell'interazione dell'utente), permettendo la comunicazione remota tramite interfacce condivise.

L'implementazione si articola nei seguenti componenti principali:

- **ServerMain**
- **RemoteGameServerImpl**
- **RemoteGameStateManagerProxy**
- **ClientMain**

Composizione del sistema

- **ServerMain** è il punto di ingresso del lato server. Si occupa della registrazione dell'oggetto remoto GameServer all'interno del registro RMI, rendendolo accessibile ai client.
- **RemoteGameServerImpl** rappresenta l'implementazione del servizio remoto GameServer. Questo oggetto gestisce le operazioni fondamentali di gioco: registrazione e disconnessione dei giocatori, aggiornamento delle posizioni e delle masse, gestione del ciclo di gioco e diffusione dello stato aggiornato a tutti i client.
- **ClientMain** rappresenta il punto di avvio del lato client. Si connette al server remoto tramite RMI, effettua il join al gioco creando un nuovo Player, e inizializza la GUI (LocalView) che mostra lo stato del gioco aggiornato in tempo reale. Un timer Swing gestisce il rendering periodico della view.
- **RemoteGameStateManagerProxy** funge da **proxy locale** per il client, permettendo l'accesso allo stato di gioco aggiornato esposto dal server remoto. Questo componente incapsula le invocazioni RMI e fornisce un'interfaccia semplificata alla view per ottenere lo stato del giocatore e degli avversari.

Comunicazione e sincronizzazione

La comunicazione tra client e server avviene tramite oggetti remoti condivisi. I client comunicano con il server esclusivamente attraverso l'interfaccia remota GameServer, richiedendo operazioni come *joinGame*, *leaveGame*, *updatePlayerState*, ecc.

Poiché la grafica è gestita localmente sul client, il server si limita a inviare lo stato aggiornato, evitando carico computazionale inutile. Ogni client ha un proprio *RemoteGameStateManagerProxy* che mantiene aggiornato lo stato visibile, interrogando periodicamente il server.

Gestione della concorrenza

Lato server, l'implementazione tiene conto della concorrenza tra più client che aggiornano simultaneamente lo stato del gioco. Sono utilizzate strutture dati sincronizzate e meccanismi di gestione dello stato centralizzato per mantenere la coerenza.