Pcd-Assigment03 - part 3 Report

Java RMI

Bedeschi Federica
0001175655 federica.bedeschi4@studio.unibo.it

Pracucci Filippo
0001183555 filippo.pracucci@studio.unibo.it

Anno accademico 2024-2025

Indice

1 Analisi **2**
1.1 Descrizione del problema 2
1.2 Visione distribuita 2

2 Design e architettura **3**

3 Conclusioni **4**

Analisi

1.1 Descrizione del problema

Agar.io è un popolare gioco multiplayer online in cui i giocatori controllano una cella circolare in un ambiente 2D. Lo scopo principale è di aumentare la propria dimensione consumando due tipi di entità:

- Cibo: piccole entità statiche distribuite casualmente, che permettono ai giocatori di aumentare la propria dimensione quando consumate;
- Giocatori: possono mangiare giocatori di dimensioni inferiori.

La partita termina quando un giocatore raggiunge 1000 unità di massa.

1.2 Visione distribuita

Seguendo un approccio distribuito abbiamo rispettato i seguenti requisiti:

- il gioco deve essere sempre attivo: i giocatori possono unirsi alla partita in ogni momento e cominciare a giocare;
- ogni giocatore deve avere una visione consistente del mondo, incluse le posizioni di tutti i giocatori e del cibo;
- il cibo deve essere generato casualmente e visibile a tutti i giocatori, e quando consumato deve essere rimosso dal sistema per tutti i giocatori;
- la condizione di fine partita deve essere controllata e imposta in maniera distribuita a tutti i giocatori.

Design e architettura

Abbiamo utilizzato il paradigma object-oriented con invocazione remota di metodi sfruttando il framework **Java RMI**. Abbiamo suddiviso il sistema in due componenti:

- `Server`: contiene il `GameStateManager` del sistema, che ha il compito di gestire il mondo di gioco ciclicamente, e aggiorna di conseguenza la `GlobalView`;
- `Client`: rappresenta un giocatore e ne gestisce la sua `LocalView`.

Il server è unico e i vari client ricevono gli aggiornamenti grazie al `GameStateManager` condiviso.
Per gestire la fine della partita abbiamo utilizzato il pattern **Observer**: è presente un *listener* per ogni client, il quale viene aggiornato dal server nel caso ci sia un vincitore.
Inoltre, in caso di errori nei client, essi subiscono una disconnessione *graceful*.

Conclusioni

Il framework **Java RMI**, a differenza di **Akka**, non permette una gestione sofisticata degli aspetti distribuiti, come il riavvio di un attore che subisce un'interruzione. Tuttavia, ci ha consentito un semplice passaggio dall'implementazione di una versione centralizzata del gioco Agar.io ad una distribuita, grazie all'utilizzo di chiamate remote di metodi su oggetti distribuiti, preservando il design object-oriented realizzato per la versione centralizzata.