Francesco Boscia e Christian Conti

Documentazione Tecnica: Applicazione "Crazy Time Multi-Player"

1. Introduzione e Scopo dell'Applicazione

L'applicazione è un simulatore multi-giocatore basato su un'architettura **Client-Server**. Lo scopo è permettere a più utenti di connettersi contemporaneamente a un server centrale, dove ogni giocatore può gestire il proprio saldo virtuale e scommettere in modo autonomo sulla ruota della fortuna.

Il sistema garantisce:

- **Multi-threading:** Ogni giocatore è isolato in un thread dedicato (CrazyWorker), permettendo sessioni di gioco indipendenti.
- **Integrità dei Dati:** L'uso di metodi sincronizzati previene la corruzione del saldo in caso di operazioni simultanee.
- **Interfaccia Asincrona:** Grazie a un thread di ascolto separato sul client, l'utente riceve aggiornamenti dal server senza bloccare l'input da tastiera.

2. Descrizione del Funzionamento

2.1 Il Server (CrazyServer e CrazyWorker)

Il server rimane in ascolto sulla porta **7777**. All'arrivo di una connessione:

1. **Accettazione:** CrazyServer accetta il socket e avvia immediatamente un nuovo thread CrazyWorker.
2. **Gestione Stato:** Ogni worker mantiene le variabili locali del giocatore (saldo, nPuntate).
3. **Ciclo di Gioco:** Il worker attende i comandi dell'utente, valida le scommesse e genera un esito casuale della ruota specifico per quel giocatore quando richiesto.

2.2 Il Client (CrazyClient e ServerListener)

Il client gestisce l'interazione con l'utente:

1. **Connessione:** Stabilisce il tunnel TCP con il server.
2. **Ascolto Attivo:** Avvia ServerListener in un thread separato. Questo thread legge i messaggi dal server (vincite, errori, risultati) e li stampa a video mentre l'utente sta ancora scrivendo la puntata successiva.
3. **Invio Comandi:** Invia stringhe formattate al server per piazzare scommesse o avviare la ruota.

3. Il Protocollo di Comunicazione (Protocollo di Applicazione)

Il protocollo è di tipo **testuale** basato su stringhe inviate tramite socket.

3.1 Sequenza Temporale dello Scambio Messaggi

Fase 1: Inizializzazione (Handshake)

- **Server → Client:** Invia il messaggio di benvenuto, il saldo iniziale (500) e la lista dei bersagli validi.
- **Scopo:** Sincronizzare l'utente sulle regole e sullo stato economico iniziale.

Fase 2: Ciclo di Puntata

L'utente può piazzare più scommesse nello stesso turno prima di far girare la ruota.

1. **Client → Server:** Invia la scommessa nel formato BERSAGLIO,CIFRA (es. 10,50).
2. **Server → Client:** Valida il bersaglio e il saldo. Se corretti, invia un messaggio di conferma e il saldo residuo.
3. **Client → Server:** Può inviare un'altra scommessa o il comando GIRA per terminare la fase di puntata.

Fase 3: Estrazione e Calcolo Vincita

1. **Server → Client:** All'invio di GIRA, il server genera l'esito della ruota tramite un array di 54 segmenti.
2. **Server → Client:** Invia il risultato (es. >>>> RISULTATO: COIN_FLIP <<<<).

3. **Server → Client:** inivia calcolo la vincita basandosi su tutte le puntate effettuate nel turno e aggiorna il saldo.

Fase 4: Chiusura

- **In caso di Saldo = 0:** Il server invia GAME_OVER e il client termina l'esecuzione.
- **In caso di comando EXIT:** Il socket viene chiuso e il thread del server viene terminato correttamente.

4. Logica della Ruota e Moltiplicatori

Il sistema simula una ruota reale con le seguenti probabilità e pagamenti:

Segmento	Quantità	Moltiplicatore
1	21	1X
2	13	2X
5	7	5X
10	4	10X
COIN_FLIP	4	20X
PACHINKO	2	50X
CASH_HUNT	2	50X
CRAZY_TIME	1	70X

5. Gestione della Concorrenza e Robustezza

Sebbene ogni partita sia indipendente, il codice implementa misure di sicurezza cruciali:

- **Metodi Sincronizzati:** I metodi processaScommessa ed eseguiEstrazione sono synchronized. Questo garantisce che le operazioni sul saldo siano atomiche e che non ci siano interferenze nei calcoli.
- **Validazione dei Bersagli:** Prima di accettare una scommessa, il server verifica che il bersaglio inserito appartenga all'elenco dei 54 spicchi reali, scartando input errati (es. "ROSSO" o "CIAO").
- **Gestione Risorse:** la gestione delle IOException assicurano che i socket vengano chiusi e la memoria liberata anche in caso di disconnessione improvvisa (es. chiusura forzata della finestra del client).