

Relazione del Progetto The “Real” Paroliere Online

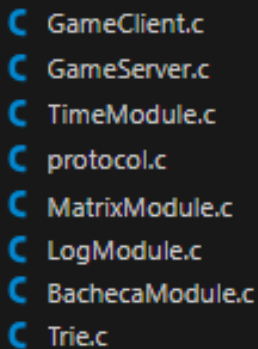
Paroli Andrea - n° 615944

1. Introduzione

Il progetto implementa un gioco multigiocatore basato sul classico "Il Paroliere". Il sistema è stato sviluppato seguendo un'architettura client-server, con gestione della comunicazione tra i client e il server tramite socket e un protocollo personalizzato come richiesto dalle specifiche del progetto. Il codice è interamente scritto in linguaggio C e utilizza strutture dati, sistemi multithread e algoritmi ricorsivi per coprire tutte le specifiche richieste dal progetto.

2. Struttura del progetto

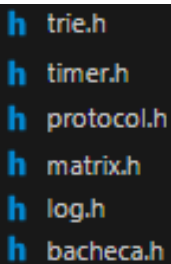
2.1 Organizzazione dei File



```
C GameClient.c
C GameServer.c
C TimeModule.c
C protocol.c
C MatrixModule.c
C LogModule.c
C BachecaModule.c
C Trie.c
```

File .c

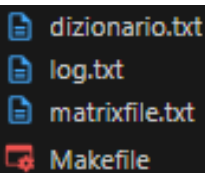
GameClient.c	- File principale (contenente main) della gestione del server
GameServer.c	- File principale (contenente main) della gestione del server
TimeModule.c	- Modulo per la gestione dello svolgimento della partita
protocol.c	- Modulo per la gestione della comunicazione fra client e server
MatrixModule.c	- Modulo per la gestione e la generazione delle matrici
LogModule.c	- Modulo per la gestione della registrazione utenti e file log
BachecaModule.c	- Modulo per la gestione della bacheca
Trie.c	- Modulo per la creazione e la ricerca sull'albero



```
h trie.h
h timer.h
h protocol.h
h matrix.h
h log.h
h bacheca.h
```

File .h

trie.h	- Fornisce funzioni e definizioni per le operazioni sull'albero
timer.h	- Fornisce funzioni e definizioni per la gestione del tempo partita
protocol.h	- Fornisce funzioni e definizioni per la comunicazione
matrix.h	- Fornisce funzioni e definizioni per le operazioni sulle matrici
log.h	- Fornisce funzioni e definizioni per le operazioni di log
bacheca.h	- Fornisce funzioni e definizioni per la bacheca



```
📄 dizionario.txt
📄 log.txt
📄 matrixfile.txt
🔧 Makefile
```

Altro

dizionario.txt	- Contiene il dizionario a cui fanno riferimento i controlli
log.txt	- File di log su cui mi salvo delle informazioni scelte
matrixfile.txt	- File facoltativo per la definizione di matrici
Makefile	- File che serve per compilare il progetto tramite comando “make”

2.2 Relazioni tra i moduli

- Il server utilizza i moduli MatrixModule, BachecaModule, LogModule, Trie e protocol per gestire il gioco e la comunicazione.

- Il client utilizza il modulo Protocol per inviare e ricevere messaggi dal server. Sfrutta anche delle funzioni definite in MatrixModule e in TimeModule.

2.3 Strutture dati principali

1. Trie:

- Utilizzato per rappresentare il dizionario delle parole valide.
- Permette una ricerca efficiente delle parole grazie alla sua struttura ad albero.

2. Matrice di gioco:

- Implementata come un array bidimensionale di caratteri (char matrix[4][4])
- Può essere importata da un file (se dichiarato) oppure generata randomicamente tramite appositi metodi

3. LogList:

- Struttura dinamica per memorizzare le parole inviate dai giocatori durante la partita e i rispettivi punteggi.
- Offre diverse funzioni, come ad esempio il controllo sulle parole duplicate

4. Bacheca:

- Array circolare (bacheca[8]) per salvare fino a otto messaggi inviati dai giocatori
- Sincronizzato con mutex (in fase di scrittura) per garantire l'accesso concorrente

5. BroadcastList:

- Struttura per salvare i socket dei client connessi, permettendo il broadcast di alcuni messaggi da parte del server
- Inizializzata quando il server si avvia, aggiornata quando un client si connette (viene aggiunto il socket) oppure quando si disconnette (il socket viene rimosso)

6. Mutex e variabili condivise:

- **game_mutex**: per sincronizzare lo stato del gioco
- **user_mutex**: per garantire la consistenza dei dati relativi agli utenti

2.4 Logica di implementazione

1. Backtracking per la verifica delle parole:

- Utilizzato per controllare se una parola proposta da un giocatore è presente nella matrice, rispettando le regole di movimento

2. Fisher-Yates Shuffle:

- Algoritmo usato per generare matrici casuali

3. Gestione della comunicazione client-server:

- Protocollo personalizzato descritto nelle specifiche del progetto

4. Gestione multithread:

- Ogni client connesso al server è gestito da un thread dedicato
- Il server crea diversi thread dedicati a diversi lavori (per esempio calcolare il tempo della partita e la gestione del gioco stesso)

5. Thread nel Sistema

- **GameClient:**
 1. *user_thread_id* – thread sempre in ascolto per gli input del client da tastiera
 2. *server_thread_id* – thread sempre in ascolto per le comunicazioni dal server
- **GameServer:**
 1. *timer_thread* – generato all'avvio del server, tiene traccia del tempo trascorso
 2. *thread_id* – generato per ogni client che si connette al server, si occupa della gestione del client

2.5 Motivazioni progettuali

- **Efficienza:** L'uso del Trie, della matrice bidimensionale e la gestione multithread garantiscono prestazioni ottimali per le operazioni più frequenti.
 - **Modularità:** La suddivisione in moduli rende il codice più manutenibile e estendibile
-

3. Processo Operativo

3.1 Server

- **Inizializzazione:**
 - Recupera e inizializza i vari parametri inseriti nel comando di avvio
 - Carica il dizionario in un Trie
 - Prepara la matrice di gioco (casuale o da file)
 - Crea thread dedicati per diverse funzioni
- **Gestione delle connessioni:**
 - Crea un thread per ogni client connesso
 - Stampa su file di log ogni utente che si registra
 - Sincronizza l'accesso alle risorse condivise tramite mutex
 - Quando il server termina, si occupa di chiudere i socket di comunicazione

- **Ciclo di gioco:**
 - Inizia una partita e stampa la matrice su console
 - Termina la partita dopo il tempo stabilito e invia i risultati
 - Stampa su file di log i risultati della partita
 - Prevede una pausa tra le partite

3.2 Client

- **Interfaccia utente:**
 - Accetta comandi come registra, login, cancella utente, matrice, parola, msg e show-msg
 - Si viene disconnessi in automatico dopo 120 secondi (AFK_TREASHOLD)
- **Gestione comunicazione:**
 - Un thread dedicato gestisce i messaggi ricevuti dal server
 - Un secondo thread gestisce i comandi da tastiera inviati dall'utente e si occupa dell'invio del messaggio al server

4. Istruzioni per la compilazione ed esecuzione

4.1 Compilazione

Per compilare il progetto si utilizza il comando:

make

Verranno generati gli eseguibili *server* e *client*.

4.2 Avvio del server

Eseguire il server con:

```
./server nome_server porta_server [--matrici data_filename] [--durata durata_in_minuti] [--seed rnd_seed] [--diz dizionario]
```

- `nome_server`: Nome o IP del server. (testato con localhost)
- `porta_server`: Porta su cui ascoltare le connessioni. (testato con 8080)
- **Opzioni:**
 - `--matrici`: File da cui caricare le matrici. (testato con matrixfile.txt)
 - `--durata`: Durata della partita (default: 3 minuti).
 - `--seed`: Seed per la generazione casuale.
 - `--diz`: File del dizionario. (testato con dizionario.txt)

4.3 Avvio del client

Eseguire il client con:

./client nome_server porta_server

- nome_server: Nome o IP del server. (testato con 127.0.0.1)
 - porta_server: Porta del server. (testato con 8080)
-

5. Fase di test

Durante lo sviluppo del progetto, è stato adottato un approccio iterativo e incrementale per il testing, testando ogni funzionalità subito dopo l'implementazione.

5.1 Tipologie di test eseguiti:

- Test funzionali (verifica che ogni funzione si comporti correttamente).
 - Test dei casi limite (es. input non validi, errori nei socket, file mancanti, cancellazione utente).
 - Debugging (correzione di errori immediati durante l'esecuzione).
-

6. Conclusioni

Il progetto "Paroliere Online" è stato sviluppato seguendo i principi di modularità, efficienza e robustezza. La gestione multithread e l'uso di strutture dati ottimizzate garantiscono un funzionamento fluido, mentre il protocollo di comunicazione è stato progettato per essere chiaro e scalabile.