PROGETTO DI RETI INFORMATICHE

Michele Meazzini 635889

1. Introduzione

Nell'escape room, l'obiettivo principale è quello di uscire dalle stanze il più velocemente possibile, raccogliendo e utilizzando gli oggetti e risolvendo enigmi. Si vince la partita quando si riesce ad uscire dalla stanza, se invece scade il timer la partita è persa. In questo caso, vi è una sola stanza, "Cucina in fiamme", dove due fratelli devono scappare prima di essere avvolti dalle fiamme di un incendio.

2. Struttura del progetto

Il progetto si compone unicamente di due file, **server.c** e **client.c** i quali includono librerie esterne, funzioni dedicate all'autenticazione degli utenti, altre dedicate alla comunicazione client server e funzioni dedicate al gioco vero e proprio.
Il progetto è costruito con l'idea di rendere facile e intuitiva l'aggiunta di nuove stanze e oggetti, basta infatti modificare le uniche tre funzioni specifiche per i vari scenari di gioco, gameplay_handler(), dedicata a gestire il corretto svolgimento delle azioni dei giocatori, init_room(), dedicata all'inizializzazione dello scenario di gioco al momento della creazione della partita e start_room_handler(), che avvia la partita.
Di seguito una lista riassuntiva delle strutture dati utilizzate e delle loro variabili:

- **User**: contiene le informazioni relative a ciascun utente registrato
- **Sessione**: contiene le informazioni relative alle partite in corso
- **Room**: scenario di gioco, ne viene allocato uno per ogni partita
- **Location**: parte di stanza che contiene oggetti e può essre bloccata da un enigma (es. cassaforte)
- **Oggetto**: contiene informazioni relative ad ogni oggetto e al suo stato
- **Enigma**: contiene la descrizione e la soluzione dell'enigma che blocca la location

Le strutture User e Sessione compongono delle liste, mentre le altre sono allocate all'interno della sessione tramite una gerarchia specifica: la sessione contiene lo scenario descritto dalla struct Room, che contiene varie Location le quali contengono vari Oggetti e possono essere bloccate da enigmi.

3. Svolgimento del gioco

I giocatori, appena connessi, vengono sottoposti ad una fase di **registrazione** e poi una fase di **login**. Se svolgono le operazioni correttamente viene stampata la **lista dei comandi** e possono avviare una partita, se invece esauriscono i tentativi il client viene chiuso (per evitare che un client malevolo blocchi su di sé l'attenzione del server per troppo tempo).
Il primo giocatore che avvia una partita con il comando **start** crea una sessione, il secondo giocatore che compie la stessa operazione va ad aggiungersi alla sessione esistente, dopodiché il timer parte e il gioco inizia.
I comandi possibili sono i seguenti: start <room>, look, look <loc | obj>, take <obj>, use <obj1> (<obj2>), objs, say <mess> (funzionalità a piacere), end.

4. Funzionalità a piacere - comando <say>
Per permettere la collaborazione dei giocatori ho deciso di introdurre un sistema di messaggi (stile chat) utilizzabile tramite il comando "**say <mess>"**. Il giocatore destinatario vedrà comparire immediatamente il messaggio sullo schermo e potrà rispondere con lo stesso comando. L'utilizzo di questo comando è necessario per risolvere l'enigma che blocca un oggetto fondamentale nello scenario presente.

5. Implementazione della connessione
La connessione che viene stabilita è di tipo **TCP**, questo perchè non è indispensabile una grande velocità, ma è necessaria l'affidabilità della trasmissione.
I messaggi vengono inviati in **formato testo** e il protocollo utilizzato prevede una lunghezza di ciascun messaggio fissata a 1KB (lunghezza del buffer), questo per evitare traffico dato dalla eventuale comunicazione della lunghezza del messaggio. Con la funzione send viene trasmesso l'intero contenuto del buffer, cosa che non ho ritenuto problematica data la velocità delle tecnologie odierne. L'unica eccezione è nella fase di registrazione e di login, dove username e password vengono trasmessi in messaggi di lunghezza variabile.

6. Server
Il server è implementato con la tecnica dell'**I/O multiplexing**, che permette una gestione facile e intuitiva di molteplici connessioni evitando di andare a creare più processi che, in un'applicazione del genere, risulterebbero eccessivi.
Il server riceve fino a 10 connessioni in attesa, numero arbitrario e facilmente modificabile nella funzione listen.
Attualmente non vi è un limite al numero di sessioni che possono essere create data la leggerezza del gioco.

7. Client
Il client è costruito per essere più leggero possibile, ogni stringa scritta dall'utente è inviata al server dove viene controllata e gestita, mentre nel client non vengono fatti controlli ad eccezione delle fasi di registrzione e login, dove lato client vengono controllati eventuali inserimenti nulli (username o password vuoti a causa di una pressione del tasto invio non intenzionale).

8. Soluzione
La soluzione prevede vari step che rappresentano l'avanzamento del gioco nel corretto ordine logico:

- Entrambi i giocatori utilizzano straccio e bottiglia per respirare meglio nel fumo
- Uno dei giocatori utilizza la chiave (il gioco non termina)
- Entrambi i giocatori utilizzano la foto e si comunicano le informazioni trovate
- Uno dei giocatori apre la cassaforte con il codice trovato al punto precedente, prende la pistola e la utilizza.

Ad ogni comando <use> viene mostrato un messaggio che suggerisce cosa fare.