Carlo Mazzanti 596385

Introduzione

Il progetto implementa il gioco della Escape Room tramite l'utilizzo di un software eseguito da un processo server ed un software che può essere eseguito da più processi client.

Si adotta il protocollo TCP per tutte le connessioni create in quanto non ci interessa uno scambio di informazioni veloce ma ci interessa l'affidabilità in quanto eventuali perdite di pacchetti potrebbe causare uno stato inconsistente dell'applicazione.

Una partita è gestita completamente lato server e l'unico compito dei client è ricavare il comando che l'utente vuole eseguire, controllare la sintassi, mandare tale comando al server utilizzando un formato prestabilito e controllare la risposta del server.

Sono stati implementati due tipi di enigmi: indovinello e utilizzo di certi oggetti. Gli indovinelli bloccano la possibilità di utilizzare il comando take con certi oggetti mentre il secondo tipo implica un certo utilizzo del comando use.
La corretta risoluzione di un indovinello offre la possibilità di raccogliere un certo oggetto.
Il corretto utilizzo di oggetti sblocca una nuova descrizione su un altro oggetto che è visualizzabile tramite il comando look. Il comando take inserirà l'oggetto nell'inventario del giocatore (se l'oggetto non è bloccato da indovinello oppure l'indovinello è stato risolto).

Il comando use richiede che il primo oggetto sia specificato sia presente nell'inventario.
Un oggetto può essere raccolto anche se bloccato da enigma di tipo utilizzo di oggetto.

Server

Il server implementa i meccanismi del gioco utilizzando i seguenti file di testo:

- accounts.txt contiene nome utente e password degli account già creati
- lista.txt contiene il numero ed i nomi delle stanze presenti sul server
- salotto.txt e test.txt che contengono le informazioni sullo svolgimento delle partite su tali stanze. test.txt contiene semplicemente una stanza creata al solo scopo di testare le funzionalità del server

Il server crea un socket e si mette in ascolto tramite il comando start (nel codice sorgente consegnato il server viene automaticamente avviato sulla porta 4242 in modo tale che utilizzando lo script i client si riescano a connettersi).

Il server si basa sull'I/O multiplexing in quanto deve gestire richieste provenienti da più client. Ogni volta che un client si connette per la priva volta al server viene creata una struttura dati chiamata sessione che contiene i seguenti dati: socket da utilizzare per comunicare a tale client, dati dell'account a cui si è fatto l'accesso, porta che il client utilizza per la messaggistica in partita (funzione a piacere) ed un intero che identifica se tale client si tratta del giocatore 1 o 2.

Le richieste che il client effettua utilizzano un protocollo di tipo text mentre la risposta del server segue un protocollo che dipende dal tipo di richiesta effettuata.

Le richieste occupano sempre REQ_SIZE byte e seguono il seguente formato:

nome_richiesta [argomento1] [argomento2]

dove gli argomenti sono opzionali o obbligatori in base a quale richiesta è stata effettuata.
Adottando un formato di dimensione fissa permette di semplificare la comunicazione tra i processi ma potrebbe generare traffico inutile: per limitare questo problema REQ_SIZE è stato dimensionato in maniera opportuna in modo tale da contenere tale fenomeno: il nome della richiesta non supera mai i 6 byte, gli argomenti non superano mai i MAX_NAME_LENGTH byte,

perciò considerando i due eventuali spazi
REQ_SIZE = 2 * MAX_NAME_LENGTH + 2.

Ogni volta che il server riceve una richiesta deve recuperare la sessione corrispondente e per farlo deve scorrere un array contenente tutte le sessioni, ciò può causare un significativo calo delle prestazioni per i client aventi sessioni memorizzate in fondo all'array. Si potrebbe evitare questa ricerca implementando il multithreading ed assegnando un thread per ogni client.

Le risposte che il server generano variano di formato e dimensione ma tutte quelle che adottano text protocol seguono tale formato:

<CODICE><DESCRIZIONE>

Dove codice esprime l'esito della richiesta sempre su due byte e descrizione contiene un dato richiesto oppure la descrizione dell'esito della richiesta. La dimensione di descrizione varia in base al tipo di richiesta in modo tale da minimizzare il traffico.

Client

Il client opera in due "modalità":

1) modalità menù dove ciclicamente preleva comandi dall'utente ed effettua richieste al server finchè non viene avviata una partita
2) modalità partita dove effettua un I/O multiplexing semplificato in modo tale da interagire con la partita ed implementare la funzionalità a piacere

Il client tenta di minimizzare le richieste inviate al server prevenendo l'invio di comandi con formato errato. Il client è stato implementato in modo tale da evitare di mandare richieste che sicuramente genereranno una risposta di errore da parte del server (per esempio impedisce di inviare comandi riservati ad una partita quando nessuna partita è stata creata oppure non permette l'avvio di una partita quando il login non è stato effettuato).

Funzionalità a piacere

La funzionalità a piacere implementata è la messaggistica in tempo reale con l'altro giocatore presente in partita. Al momento del login il client comunica al server la porta su cui accetterà messaggi dall'altro giocatore in un'eventuale nuova partita. Una partita permette la collaborazione tra al massimo 2 giocatori. All'avvio della partita il server cerca prima la presenza di un'altra partita libera in corso sulla stessa stanza, se la trova al client viene associata tale partita, altrimenti ne crea una nuova. Se viene creata una nuova partita allora il client sarà il giocatore 1 ed avrà il compito di creare un socket listener che sarà in attesa della connessione del giocatore 2, una volta accettata la connessione il listener viene chiuso. Se si partecipa alla partita già esistente allora il client richiederà al server la porta su cui contattare l'altro giocatore ed inizializza la connessione. Tale connessione oltre per la messaggistica verrà utilizzata anche per la sincronizzazione dei client all'interno della partita. Il protocollo adottato è di tipo text e segue lo stesso formato adottato per le risposte del server a differenza del fatto che il campo codice occupa un solo byte. Ogni messaggio occupa 1 + MAX_DESCRIPTION_LENGTH byte dove il primo byte è riservato al tipo di comunicazione effettuata (essenziale per distinguere un normale messaggio dalla comunicazione della terminazione della partita). Tale sincronizzazione è necessaria poiché il server comunicherà solo a chi effettua l'ultima azione della partita la terminazione della stessa, perciò chi riceve per primo la comunicazione della terminazione della partita (tempo scaduto, partita vinta) ha il compito di comunicare all'altro giocatore tale informazione. Se un giocatore termina la partita tramite il comando end allora comunicherà allo stesso modo la fine della partita all'altro giocatore.

Insieme ai file del progetto è presente il file informazioni.txt dove sono riportati nel dettaglio tutti i formati e dimensioni di richieste e risposte ed il significato di tutti i codici adottati nel progetto.