PROGETTO DI **RETI INFORMATICHE**
A.A 23-24 (615550 - *Lorenzo Valtriani*)

1. *Introduzione generale*

L'applicazione contiene al suo interno solo una singola room giocabile, ma l'intero ecosistema dell'applicazione stessa rende facile l'aggiunta e la modifica di altre stanze, che sono tutte descritte all'interno della struttura dati ***desc_room***, che fa uso anche delle strutture dati ***desc_location*** e ***desc_obj*** (i cui campi sono commentati all'interno del file game.h in cui sono definite).
Il gioco prevede due tipologie di enigmi: quiz a risposta multipla e l'utilizzo di oggetti in un determinato ordine prefissato.
E' stato aggiunto un comando extra oltre a quelli richiesti: *drop* (rimuovere un oggetto dalla borsa).
La funzionalità a scelta è rappresentata da un altro utente (other.c) che invece di giocare può aumentare o diminuire il tempo della partita di uno specifico utente che sta giocando in quello specifico momento.
Il livello di trasporto scelto è il TCP in quanto era necessario trasferire dati in modo affidabile.
Lo scambio dei dati tra client e server è implementato utilizzando la modalità binary per l'invio/ricezione della dimensione di un unico messaggio, il cui invio è implementato poi attraverso la modalità text, questo perché la maggior parte dei dati da inviare sono stringhe, quindi risulta più naturale ed intuitivo, anche in ottica di debug.
Questo metodo però ha come conseguenza un consumo di banda superiore rispetto ad utilizzare la modalità binary.

2. *Implementazione server*

L'applicazione è maggiormente I/O bound, poiché il tempo che l'utente impiega a ragionare sui prossimi comandi da utilizzare o per risolvere gli enigmi, è notevolmente superiore rispetto al tempo di computazione necessario per l'esecuzione di ogni richiesta. Per questa ragione il server è basato su I/O multiplexing. Oltre al motivo precedentemente descritto l'overhead per l'I/O multiplexing, che è per lo più dato dalla gestione delle socket pronte, implementata usando una lista ordinata per fd in modo decrescente (al posto di un array), così da avere come primo descrittore sempre quello con fdmax. Questo overhead è comunque inferiore a quello derivato dai vari cambi di processo in un sistema multiprocesso. Il problema dell'overhead potrebbe però essere superato se utilizzassimo un sistema multithreading al posto di un sistema multiprocesso.
Il server è passivo, e quando un socket di comunicazione risulta pronto in lettura, secondo il protocollo, in primo luogo effettua una receive da quella socket, analizza il messaggio ricevuto dal client, esegue le operazioni relative a quel messaggio, e infine invia la risposta al client.
La risposta del server al il client è descritta dalla struttura ***desc_resp*** (tcp.h), i cui campi principali sono: ***`expected`***, rappresenta il tipo di messaggio che il server si

aspetta di ricevere dal client la volta dopo; ***response,*** di dimensione ***dim_response*** rappresenta la risposta testuale che il client deve visualizzare, può essere vuota se il campo ***notify_code*** risulta valido (diverso da -1); ***seconds*** rappresenta il numero di secondi che rimangono al giocatore; ***token*** indica, in risposta al comando di *start*, il numero di token da raccogliere per vincere e per le successive risposte la presenza di un nuovo token ottenuto (se 1).

```
typedef struct desc_resp {
    enum TypeMessage expeted;       /* ciò che il server si aspetta al prossimo messaggio inviato dal client */
    int dim_response;               /* dimensione della risposta testuale */
    char response[MAX_DIM_RESP];    /* messaggio che il server comunica al client */

    int token;                      /* usato la prima volta per dire al client quanti token sono necessari, succesivamente per segnalare
    int seconds;                    /* usato per dire al client i secondi rimanenti */

    int notify_code;                /* contiene il tipo di notifica che l'utente dovrà visualizzare */
    bool status;                    /* notifica all'utente se la cosa è andata a buon fine */
} desc_resp;
```

3. *Implementazione client*

Il client è una semplice applicazione che invia comandi al server senza avere informazioni sulla specifica room, quindi spostando la complessità sul server.
Il client comunica con il server attraverso alcuni messaggi di tipo ***desc_msg*** (tcp.h) ogni volta che il giocatore inserisce un comando o risponde ad un enigma; il descrittore è formato da alcuni campi: ***type***, indica se il messaggio rappresenta un comando oppure una risposta testuale (di un enigma); ***command***, contiene il comando inserito dal giocatore o la risposta all'enigma; ***operand_1*** e ***operand_2***; contengono, se usati, gli operandi del comando usato dal giocatore; ***num_operand*** indica il numero di operandi inseriti dal giocatore.
Dopo aver inviato il messaggio al server, rimane in attesa della risposta di quest'ultimo (il server invece in questo momento non fa attesa attiva), poi controlla se i secondi rimanenti sono esattamente zero, se lo sono, ci possono essere due motivi: il giocatore ha inviato il comando di *end* o il tempo a disposizione è scaduto, in questi casi si chiude la connessione con il server e l'applicazione termina la sua esecuzione. Nel caso in cui il tempo non sia terminato, si controlla la presenza di un nuovo token, e nel caso, si controlla se il giocatore ha pure vinto; altrimenti il ciclo ricomincia mostrando la risposta del server, e il programma, si mette in attesa di una nuova interazione del giocatore.

```
typedef struct desc_msg {
    enum TypeMessage type;          /* RESPONSE_TEXT, COMMAND, SHADOW */
    char command[MAX_DIM_PARAM];    /* definisce l'operazione che il client ha richiesto o la risposta testuale */
    char operand_1[MAX_DIM_PARAM];  /* definisce il primo operando dell'operazione, se lo ha */
    char operand_2[MAX_DIM_PARAM];  /* definisce il secondo operando dell'operazione, se lo ha */
    int num_operand;                /* definisce il numero di operandi (0, 1, 2) */
} desc_msg;
```

4. *Implementazione client ombra*

L'utente ombra è un utente che invece di giocare, influenza la partita di altri giocatori che stanno giocando in quel momento specifico.
Ha a disposizione tre comandi principali: *list*, richiede al server una lista di giocatori che in quello specifico momento stanno giocando una partita; + *id_utente secondi*, permette di aumentare il tempo rimanente alla partita di quel determinato utente;
- *id_utente secondi*, esattamente l'opposto.
La comunicazione tra il client ombra e il server avviene esattamente come nel client, usando quindi lo stesso protocollo.