UNIVERSITÀ DI PISA

CORSO DI LAUREA IN INGEGNERIA INFORMATICA
SPECIFICHE PROGETTO A.A. 2023/2024

Reti Informatiche, cod. 545II, 9 CFU

Prof. Giuseppe Anastasi, Ing. Francesco Pistolesi

Si richiede di progettare un'applicazione distribuita che implementi il gioco dell'*escape room*, in uno scenario digitale.

1. VISIONE D'INSIEME

L'Escape Room è una forma di intrattenimento interattiva e stimolante che sfida i partecipanti a risolvere enigmi all'interno di uno spazio tematico, con l'obiettivo di "fuggire", completando una missione entro un limite di tempo predeterminato. L'esperienza è progettata per coinvolgere i giocatori in un'atmosfera avvincente, richiedendo creatività, logica e collaborazione per superare le sfide presenti nella stanza.

Il nucleo del gioco è costituito dagli enigmi, che possono variare da rompicapi logici a misteri complessi che richiedono soluzioni creative. I partecipanti esplorano lo spazio, cercano indizi, combinano oggetti e risolvono enigmi per progredire nella trama e raggiungere l'obiettivo di uscire dalla *room*.

La componente collaborativa è fondamentale, poiché i giocatori devono unire le proprie abilità, comunicare in modo efficace e sfruttare le competenze individuali per superare le sfide.

L'Escape Room può essere strutturata su una varietà di temi, da scenari di suspense a contesti storici, garantendo un'esperienza diversificata e coinvolgente per i partecipanti. L'ambientazione può quindi variare, offrendo ai giocatori l'opportunità di immergersi in mondi fantastici, avventure storiche o situazioni più moderne. La tensione del tempo, presente con un conto alla rovescia, aggiunge un elemento di suspense e urgenza al gioco.

Per usare l'applicazione la prima volta, un utente deve effettuare un'operazione di *signup*, contattando il server per creare un account costituito da username e password. Successivamente, l'utente può effettuare il *login*, accedendo così al menu dell'applicazione. Da quel momento, l'utente è online fino al *logout*.

Dopo il login, il giocatore può visualizzare gli **scenari di gioco**[1] disponibili, e selezionarne uno. Dopo la selezione, il gioco inizia. Uno scenario consiste in una room, all'interno della quale vi sono varie locazioni.

[1] È sufficiente la presenza di uno scenario. Più scenari possono essere semplicemente creati combinando gli enigmi o la loro successione.

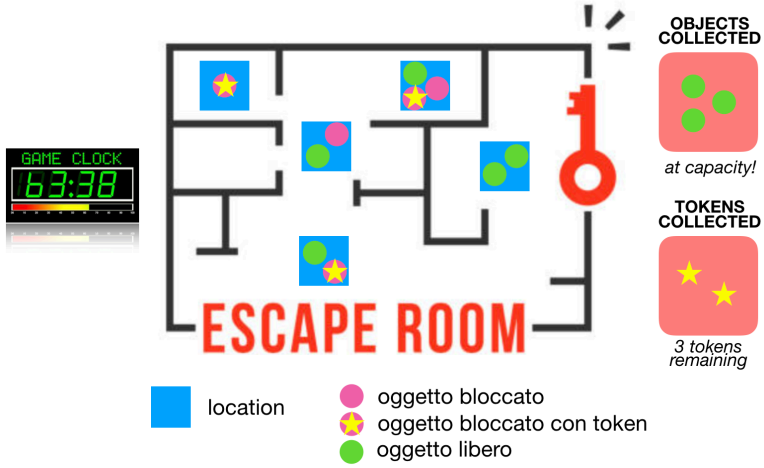

Figura 1: Rappresentazione grafica di una escape room.

La Fig. 1 illustra il modello di escape room considerato in questo contesto. Come mostra la figura, ogni **locazione** (*location*) contiene uno o più oggetti, alcuni dei quali sono liberi, cioè si possono raccogliere o usare con altri oggetti. In una location, possono anche esservi oggetti bloccati, i quali possono essere raccolti, oppure raccolti e usati solo dopo aver risolto un enigma. Al giocatore non è noto se un oggetto può essere usato da solo, oppure quale sia l'altro oggetto col quale questo deve essere usato. Il giocatore può raccogliere un numero limitato di oggetti. Quando il limite massimo viene raggiunto — come mostra, per esempio, la parte in alto a destra della Fig. 1 — il giocatore deve depositare uno degli oggetti in suo possesso, per poterne raccogliere un altro. Un **enigma** può essere una domanda a risposta multipla, un indovinello con numero limitato di risposte concesse, il completamento di un proverbio, oppure la raccolta di oggetti presenti nelle stanze in un determianato ordine, o il loro utilizzo con altri oggetti in un determinato ordine. È richiesta la presenza di almeno due tipologie di enigma nell'escape room da progettare. Le due tipologie possono essere scelte fra quelle elencate precedentemente, oppure altre. La risoluzione di un enigma sblocca l'oggetto associato, che può quindi essere raccolto e assegnare un **token** al giocatore, oppure raccolto e far guadagnare del tempo bonus. Il gioco termina quando il giocatore riesce a risolvere tutti gli enigmi che assegnano token entro il tempo a disposizione, oppure quando scade il tempo. Il conto alla rovescia parte nel momento in cui il gioco ha inizio.

Questo scenario di base va implementato come descritto. I dettagli dei comandi specifici sono forniti nel Paragrafo 2. Occorre poi implementare un'ulteriore funzionalità, a piacere, che introduca la comunicazione fra il giocatore e un altro host, il cui ruolo e interazione col giocatore possono essere liberamente scelti.

Per l'implementazione dell'intero progetto, scegliere i protocolli di comunicazione, le strutture dei pacchetti, le modalità di scambio dei dati (text o binary), e la tipologia di server (iterativo, concorrente, o basato su I/O multiplexing) dipendentemente da aspetti che si ritengono congrui all'applicazione descritta. Le considerazioni fatte per prendere tutte queste decisioni devono essere spiegate in una documentazione da consegnare assieme ai sorgenti.

2. DETTAGLI IMPLEMENTATIVI

L'applicazione distribuita da sviluppare deve implementare quanto descritto nel Paragrafo 1. Le scelte progettuali devono essere spiegate in una relazione di **non più di 2 pagine**, font Arial, dimensione 12 pt, usando anche figure[2] e schemi intuitivi. La documentazione deve contenere anche la descrizione della funzionalità a piacere, con i relativi dettagli implementativi.

[2] Le figure possono essere semplicemente realizzate a mano, e poi fotografate e inserite nella documentazione.

Realativamente all'intero progetto, nella relazione devono essere messi in luce, in modo critico, potenziali pregi e difetti delle scelte fatte in termini di quantità di traffico generato, possibili bottleneck, scalabilità, e così via. Documentazioni contenenti solo le scelte fatte, senza un'analisi critica, non saranno considerate sufficienti: dovranno essere integrate in sede di orale. Se la documentazione consegnata contiene più di due pagine, la valutazione del progetto avverrà considerando solo le prime due.

2.1 SERVER

Il **server** è mandato in esecuzione sull'host come segue:

```
./server <porta>
```

dove **<porta>** è la porta associata al peer.

Appena mandato in esecuzione, il processo **server** mostra a video una guida dei comandi. Dopo l'inserimento di un comando da standard input, oppure dopo la ricezione di un messaggio dalla rete da parte dei client, il server mostra a video, passo passo, cosa sta facendo, con un formato a piacere[3].

I comandi accettati dal server, da standard input, sono:

start *port*
avvia il server sulla porta *port*. Dopo l'avvio, il server si mette in attesa di connessioni da parte dei giocatori.

stop
il server richiede si disconnette dal network. Gestire opportunamente la chiusura delle connessioni. Il server non può essere arrestato se c'è almeno una partita in corso.

Un esempio di output mostrato dal server dopo l'esecuzione è il seguente:

```
****************************** SERVER STARTED ********************************

Digita un comando:

1) start <port> --> avvia il server di gioco
2) stop --> termina il server
******************************************************************************
```

Il server deve poi essere in grado di processare le richieste generate dai comandi del client, descritti nel seguente paragrafo. Quando un giocatore seleziona uno scenario, il server può anche mandarlo in esecuzione senza caricarlo da file, ma semplicemente utilizzare un'istanziazione basata sull'esecuzione di codice.

2.2 CLIENT

Un **client** è mandato in esecuzione sull'host[4] come segue:

```
./client <porta>
```

All'avvio, il client mostra un menu testuale con una breve guida ai comandi e i codici delle room.

---

[3] Si può pensare che il peer lavori in modalità *verbose*, e informi l'utente continuativamente su ciò che sta facendo.

[4] Per semplicità, server e client sono eseguiti sullo stesso host. Per questo, il comando di avvio del client non necessita di un parametro per specificare l'IP del server (che sarà 127.0.0.1).

I comandi accettati dal client sono i seguenti:

`start room`
Avvia il gioco nella stanza il cui codice (*room*) è specificato come parametro. Il codice è un numero intero.

`look [location | object]`
Fornisce una breve descrizione della stanza con le sue locazioni. Ogni locazione è contraddistinta da un nome. Se il comando look è digitato includendo il parametro *location* (cioè, il nome di una location valida, il comando stampa una descrizione più dettagliata della locazione specificata e gli eventuali nomi degli oggetti ad essa associati. Se si specifica un oggetto (*object*), il comando descrive l'oggetto. La descrizione dell'oggetto può cambiare se questo era bloccato e poi viene sbloccato dalla risoluzione di un enigma. Nelle descrizioni, evidenziare i nomi delle locazioni e degli oggetti con la seguente notazione: **oggetto** ++locazione++.

Esempi di output del comando:

```
> look
Sei in una biblioteca. Sul ++tavolo++ c'è un **libro** molto grande, vicino a un
paio di **occhiali**. Di fronte a te, c'è una ++mensola++ con un **portacenere**
e una **clessidra**. Alla tua destra c'è un ++mobiletto++.
```

```
> look mobiletto
Il mobiletto è un secretaire in legno scuro. Ha una piccola **serratura** sullo
sportellino superiore. Poi c'è uno **sportello** grande in basso, ma questo è
privo di serratura.
```

```
> look libro
Questo libro è polveroso e ha la copertina molto rovinata. È chiuso da un
piccolo **lucchetto**.
```

`take object`
Consente al giocatore di raccogliere un oggetto presente nella stanza corrente, il cui nome (*object*) è specificato come parametro. Se l'oggetto è bloccato, il giocatore riceve un enigma da risolvere. Quando il giocatore risolve l'enigma, l'oggetto si sblocca e, un'ulteriore *take* sullo stesso oggetto comporta la raccolta dell'oggetto. Se l'enigma ha un numero di tentativi limitato e il giocatore li esaurisce, il gioco termina.

Esempio di enigma:

```
> take libro

Oggetto **libro** bloccato. Devi risolvere l'enigma!
Qual è l'animale che al mattino cammina a quattro zampe, nel pomeriggio a due, e
di sera a tre? Digita la risposta
_
```

Altro esempio di enigma:

```
> take chiave

Oggetto **chiave** bloccato. Devi risolvere l'enigma!
Completa la sequenza: 2, 4, 12, 14, ?
_
```

`use object1 [object2]`
Permette al giocatore di utilizzare un oggetto (*object1*) precedentemente raccolto. Se il comando contiene anche il parametro *object2*, l'oggetto *object1* viene usato con l'oggetto *object2*. L'oggetto *object2* può essere stato precedentemente raccolto, oppure no (se bloccato da un enigma).

Esempio di utilizzo del comando ed evoluzione del gioco:

```
> use chiave lucchetto
Serratura sbloccata!
> use serratura
Hai aperto il libro!
> look libro
Nella prima pagina c'è un grande disegno in cui si può leggere chiaramente il
numero 18. Ci sono poi delle **scritte indecifrabili**.
```

objs
Mostra all'utente l'elenco degli oggetti raccolti fino a quel momento.

Esempio di esecuzione:

```
> objs
chiave
portacenere
lucchetto
libro
```

end
Termina il gioco e la connessione con il server.

Dopo l'esecuzione di ogni comando, al giocatore è mostrato il numero di token raccolti, mancanti, e il tempo rimanente. Se il giocatore digita un comando in modo errato, riceve su standard output un breve aiuto su come digitare correttamente il comando rilevato.

REQUISITI

- I dati sono scambiati tramite **socket**. Se non si definisce un formato di messaggio, prima di ogni scambio, il ricevente deve essere informato su **quanti byte** leggere dal socket.

- Per gestire le varie informazioni, è possibile usare **strutture dati a piacere**. Gli aspetti che non sono dettagliatamente specificati in questo documento possono essere implementati liberamente.

- Usare gli **autotools** (comando **make**) per la compilazione del progetto.

- Il codice **deve essere indentato e commentato** in ogni sua parte: significato delle variabili, processazioni, ecc. I commenti possono essere evitati nelle parti banali del codice.

- Il **salvataggio** delle informazioni su file è opzionale. Nel caso si decida di non implementarlo, tali informazioni sono mantenute in memoria.

- Va prodotta una **documentazione di 2 pagine** (font Arial, size 12 pt) in cui giustificare, in maniera critica, le scelte fatte, e i formati di strutture dati e messaggi (ove introdotti).

- Scrivere uno **script che compili il progetto, mandi in esecuzione il server e due client**, su finestre diverse del terminale. Fare in modo che ci sia almeno uno scenario di gioco sul server.

CONSEGNA

Il progetto deve essere caricato sul portale elearn.ing.unipi.it (sulla pagina del corso 2023/2024), non oltre le 72 ore che precedono il giorno dell'esame, usando le credenziali di Ateneo per l'accesso. Per ogni appello, sarà creato un nuovo slot per le consegne.

VALUTAZIONE

Il progetto è visionato e valutato prima dello svolgimento dell'esame, eseguendolo su sistema operativo **Debian 8**, distribuzione usata durante il corso. Testare sempre il codice (compilare ed eseguire) su una macchina Debian 8 prima della consegna. Durante l'esame, può essere richiesta l'esecuzione del programma, potenziali modifiche, e saranno fatte domande sia sul codice che sulle scelte fatte.

La valutazione del progetto prevede le seguenti fasi:

1. **Compilazione del codice**
   Il client e il server vanno compilati con opzione **`-Wall`** che mostra i vari warning. Non vi dovranno essere warning o errori. L'opzione **`-Wall`** va abilitata anche durante lo sviluppo del progetto, interpretando i messaggi forniti dal compilatore. Se il progetto non compila, non è valutabile;

2. **Esecuzione dell'applicazione**
   In questa fase si verifica il funzionamento dell'applicazione e il rispetto delle specifiche;

3. **Analisi del codice sorgente**
   Può essere richiesto di spiegare parti del codice e apportarvi semplici modifiche.

FUNZIONI DI UTILITÀ

Nel caso si utilizzino i file, una documentazione di alto livello delle funzioni necessarie per leggere e scrivere file a blocchi possono essere visualizzate al seguente indirizzo:

https://www.programiz.com/c-programming/c-file-input-output