Progetto Esame Programmazione Procedurale a.a. 2022/2023 - "PhalsoPhobia: Un gioco in C"

Canali per le domande: ricevimento, email: francesco.santini@unipg.it, Telegram: @safran

7 Dicembre 2022

Si realizzi un programma in linguaggio C che consista dei seguenti tre file (utilizzare "obbligatoriamente" questi tre nomi nel progetto, che fanno comunque già parte del template Github):

- `main.c` contiene solo la definizione della funzione *main()*.
- `gamelib.c` contiene le definizioni delle funzioni che implementano il gioco.
- `gamelib.h` contiene le dichiarazioni delle funzioni definite in `gamelib.c` (solo quelle non *static*) e le definizioni del tipo delle strutture dati utilizzate in `gamelib.c`.

La storia. Un team di investigatori del paranormale viene contrattualizzato per raccogliere prove sulle presenze che si aggirano in luoghi infestati. Riusciranno a capire di che fantasma si tratta prima di impazzire?[1]

main.c. Questo file contiene solo la funzione *main()*, il cui compito è stampare un menu di scelte verso il giocatore ed aspettare il suo comando. Le possibili scelte sono: *1)* imposta gioco, *2)* gioca, *3)* termina gioco. *Suggerimento*: utilizzare un ciclo *do...while* per stampare il menu (dato che deve essere stampato per lo meno una volta), leggere la scelta del giocatore da tastiera, e con uno *switch* eseguire i comandi per effettuare la scelta richiesta. In caso il comando non sia 1-2-3, stampare un messaggio al giocatore che il comando è sbagliato, e poi ristampare il menu. Eseguire controlli simili su input anche per tutte le altre letture di scelta da tastiera che si trovano nel gioco. Nel caso la scelta sia 1, 2, o 3, chiamare la corrispondente funzione definita in *gamelib.c*, cioè *imposta_gioco()*, *gioca()*, e *termina_gioco()*. Si può impostare il gioco più volte di fila prima di combattere (liberare ogni volta tutta la memoria dinamica allocata precedentemente). Non si può combattere se prima non è stato impostato il gioco. Una volta che il combattimento è finito, si torna sempre a questo menu, dal quale è poi possibile uscire premendo 3.

[1] Esempio dell'originale a cui PhalsoPhobia è ispirato (QDSS): `https://www.youtube.com/watch?v=AkMMvXNG754&t=118s`.

gamelib.h. Questo file deve contenere solamente le dichiarazioni delle tre funzioni introdotte precedentemente, e le definizioni dei tipi utilizzati nella libreria, cioè i tipi *struct Giocatore*, *struct Zona_mappa*, *enum Tipo_oggetto_iniziale* , *enum Tipo_oggetto_zona*, *enum Tipo_zona* e *enum Tipo_prova*:

- La *struct Giocatore* contiene *i)* un array *nome_giocatore* di caratteri che rappresenta il nome del giocatore, *ii)* un campo *unsigned char sanita_mentale*, *iii)* un campo che memorizza la posizione sulla mappa (*struct Zona_mappa* posizione*) contenente il puntatore alla zona dove si trova il giocatore, e *iv)* un array *zaino* di 4 posizioni, ciascuna di tipo *unsigned char*: una posizione dell'array può memorizzare il valore che identifica l'oggetto *enum Tipo_oggetto_iniziale*, *enum Tipo_oggetto_zona*, *enum Tipo_prova* corrispondente all'oggetto/prova presente in quello slot dello zaino. Da notare che uno slot può essere anche "vuoto" non presente in nessuno dei tre enum che possono trovarsi in uno slot (aggiungere a piacere). Per esempio, se *zaino[1]* vale 1, e 1 rappresenta il *coltello*, allora nella seconda posizione dello zaino si trova il coltello.
- La *struct Zona_mappa* contiene i seguenti campi: *enum Tipo_zona zona*, *enum Tipo_oggetto_zona oggetto_zona*, *enum Tipo_prova prova* e *struct Zona_mappa* prossima_zona* ovvero un puntatore che punta alla prossima zona della mappa.
- *enum Tipo_oggetto_iniziale* deve rappresentare i valori $\{EMF, spirit\_box, videocamera, calmanti, sale\}$.
- *enum Tipo_oggetto_zona* deve rappresentare i valori $\{adrenalina, cento\_dollari, coltello, nessun\_oggetto\}$.
- *enum Tipo_zona* i cui valori possono rappresentare $\{caravan, cucina, soggiorno, camera, bagno, garage, seminterrato\}$.
- *enum Tipo_prova* i cui valori possono rappresentare $\{prova\_EMF, prova\_spirit\_box, prova\_videocamera, nessuna\_prova\}$.

gamelib.c. Questo file rappresenta il nucleo del progetto e contiene tutte le definizioni di funzione necessarie per il gioco. Tutte le funzioni che non sono *imposta_gioco()*, *gioca()*, e *termina_gioco()* devono essere definite con lo specificatore *static*, dato che non devono essere visibili all'esterno di questo file.

Funzioni da implementare in gamelib.c. Le funzioni minime da implementare nel progetto sono:

- *imposta_gioco()*. Questa funzione chiede come prima cosa di inserire da tastiera il numero di giocatori, che può variare da minimo 1 a massimo 4. Ogni giocatore è rappresentato da una *struct Giocatore*, che viene creta in memoria dinamica. L'insieme dei giocatori è rappresentato da un array di 4 *Struct Giocatore** (*NULL* se il giocatore non partecipa al gioco). La sanità mentale inizia da 100 per ogni giocatore e potrà scendere durante il gioco, il nome viene inserito da tastiera. Inoltre viene richiesto il livello di difficoltà tra *dilettante*, *intermedio*, *incubo*. Questo livello serve per calcolare alcune probabilità relative alle difficoltà del gioco (vedere nel seguito). Dopo questa fase di inizializzazione di tutti i giocatori, vengono generati a caso un numero di oggetti pari al numero di giocatori partecipanti. Utilizzare un numero casuale per generare un oggetto; tutti gli oggetti hanno la stessa probabilità di essere generati, da scegliere in *enum Tipo_oggetto_iniziale*, ****inizio modifica**** ma almeno un oggetto per raccogliere una prova deve essere assegnato ad almeno un giocatore (altrimenti non si può terminare il gioco se non si può raccogliere per lo meno una prova e portarla al caravan). ****fine modifica**** A questo punto, si seleziona sempre in ordine casuale un giocatore, che può scegliere l'oggetto da inserire nel proprio zaino.

  Dopodiché si deve generare la mappa di gioco, che è costituita da una lista di *struct Zona_mappa*. Per questo c'è quindi bisogno di due puntatori globali per memorizzare il primo e l'ultimo elemento della

lista (*struct Zona_mappa* pFirst* e *struct Zona_mappa* pLast*). La mappa è implementata da una *lista circolare*: il campo *prossima_zona* dell'ultima zona contiene l'indirizzo della prima zona. In questo modo il giocatore che si trova sull'ultima zona e avanza si troverà di nuovo sulla prima zona. Da notare che il *caravan* non fa parte di questa lista, ma è una zona a parte.

Per la creazione della mappa, si lascia la possibilità all'utente (supponiamo sia un "game master" in questo caso) di richiedere tramite un menu le seguenti quattro funzioni:

1) Inserimento di una zona **al termine** della lista (funzione *inserisci_zona()*). Essa crea la nuova zona in memoria dinamica (*malloc()*), la inserisce nella lista modificando il valore del puntatore *prossima_zona* dell'ultima terra della lista con il risultato della *malloc()*. Il tipo della zona (*enum Tipo_zona*) così come l'oggetto trovato all'interno della zona vengono generati casualmente: tutte le zone sono equiprobabili (eccetto *caravan* che non può essere mai aggiunta alla lista), *nessun_oggetto* ha probabilità 40% mentre gli altri oggetti in *enum Tipo_oggetto_zona* hanno pari probabilità. Infine, il campo *enum Tipo_prova prova* verrà inserito con una certa probabilità tutte le volte che un giocatore arriverà in una data zona (vedere funzione *avanza()* sotto). Ripercorrendo le stesse zone sarà quindi possibile trovare prove differenti.
2) Cancella l'ultima zona inserita nel percorso (*cancella_zona()*). Ricordarsi di liberare la memoria occupata dalla zona cancellata. Ricordarsi di trattare il caso particolare in cui nella lista non è presente nessuna zona.
3) Stampa i campi di tutte le zone create fino a quel momento (*stampa_mappa()*).
4) Fine della creazione della mappa: *chiudi_mappa()*. Ci si ricorda che la creazione della mappa è terminata (per esempio settando una variabile globale e statica da 0 a 1). Questa variabile viene controllata quando si chiama *gioca()*, per controllare che il gioco sia stato in effetti completamente impostato. Infine si esce anche dalla funzione *imposta_gioco()*.

- *gioca()*. Si controlla se il gioco sia stato impostato correttamente e poi, in caso affermativo, si passa alla fase di gioco vera e propria, le cui funzioni sono riportate di seguito. Tutti i giocatori iniziano il gioco sulla prima zona della lista creata (che non è *caravan*).

- *termina_gioco()* si termina il gioco salutando i giocatori.

Il gioco è strutturato in turni. Il giocatore che muove in un dato turno è scelto a caso, anche se prima di poter rigiocare un turno tutti i rimanenti giocatori devono aver giocato il loro; ad esempio, i giocatori possono giocare nell'ordine 2-4-1-3, e poi si riparte, sempre con un nuovo ordine casuale (2-4-4-1-3 invece non è possibile). Durante il turno di un giocatore, le funzioni che possono essere chiamate per giocare sono:

- *torna_caravan()*: il caravan è la base d'appoggio per i giocatori: può essere visto come una zona speciale e ne esiste sempre una ed una sola: non può essere quindi generata un'altra zona caravan durante la creazione della mappa. Con questa funzione il giocatore ritorna automaticamente al caravan: le prove eventualmente raccolte dal giocatore verranno lasciata nel caravan, marcandole come trovate. ****inizio modifica**** Al posto della prova, nello zaino comparirà uno dei rimanenti oggetti tra *EMF*, *spirit_box*, *videocamera* che non è ancora stato distribuito ai giocatori (altrimenti risulterebbe impossibile raccogliere tutte e tre le prove e concludere il gioco) ****fine modifica**** . Alla fine il giocatore verrà posizionato sulla prima zona della lista e il turno passerà al prossimo giocatore. La presenza del fantasma nella stessa zona impedisce di ritornare al caravan, e quindi questa funzione non potrà essere chiamata.

- *stampa_giocatore()*: stampa i valori di tutti i campi del giocatore.

- *stampa_zona()*: stampa i valori di tutti i campi della zona dove si trova il giocatore.
- *avanza()*: il giocatore avanza sulla zona successiva della mappa. Quando un giocatore avanza e arriva in una certa zona, il campo *enum Tipo_prova prova* di questa zona viene aggiornato in modo casuale: *nessuna_prova* 40%, mentre *prova_EMF*, *prova_spirit_box*, *prova_videocamera* tutte con 20% ciascuna. Ogni giocatore può avanzare una volta sola per turno.
- *raccogli_prova()*: se in una stanza si trova una certa prova, per esempio *prova_spirit_box*, è possibile raccoglierla solo se il giocatore è in possesso dell'oggetto corrispondente, in questo caso *spirit_box*: nello zaino, *spirit_box* viene sostituito automaticamente da *prova_spirit_box*. Ogni volta che un giocatore raccoglie una prova, c'è una probabilità dipendente dal livello di difficoltà impostato che il fantasma appaia (scegliere queste probabilità). Se il fantasma appare, tutti i giocatori in quella zona diminuiscono la loro sanità mentale di un numero dipendente dalla difficoltà del gioco selezionata (scegliere queste quantità). La probabilità di apparizione del fantasma incrementa con la prova raccolta: alla prima prova raccolta la probabilità sarà minore che alla seconda, e la seconda minore rispetto alla terza (scegliere questi incrementi).
- *raccogli_oggetto()*: in una stanza con un oggetto è possibile raccogliere l'oggetto, sempre se c'è posto nello zaino.
- *usa_oggetto()*: oltre agli oggetti per raccogliere le prove, è possibile utilizzare uno degli altri oggetti presenti nello zaino. Questi altri oggetti quando usati hanno degli effetti speciali (vedere di seguito). Quando si usa un oggetto, esso scompare dallo zaino.
- *passa()*: il giocatore cede il turno al prossimo giocatore.

Se tutte e tre le prove *prova_EMF*, *prova_spirit_box*, *prova_videocamera* vengono marcate come trovate (lasciate quindi nel caravan), allora i giocatori (sopravvissuti) hanno vinto la partita. Se invece la sanità mentale di un giocatore scende a 0 o sotto, questo giocatore non parteciperà più al gioco. Se tutti i giocatori scendono sotto 0, la partita è persa.

Di seguito l'effetto degli oggetti:

sale: se si usa il sale , se il fantasma apparirà non provocherà nessun decremento di sanità mentale.

calmanti: aumenta di 40 punti la sanità mentale.

100 dollari: se si usano i dollari, al posto dei dollari nello zaino compare un oggetto a caso tra *calmante* e *sale*.

coltello: se si usa il coltello e la propria sanità mentale è sotto 30, si uccidono tutti gli altri membri nella zona in cui ci si trova (se ci sono).

adrenalina: se si usa adrenalina si può avanzare una volta in più (chiamare *avanza()*).

Infine, per ogni azione compiuta da un giocatore, c'è una probabilità del 20% che la sanità mentale decresca di 15 punti.

Al termine di una partita, vinta o persa, si può ritornare al menu iniziale e reimpostare il gioco e rigiocare. Ricordarsi prima di deallocare tutta la memoria dinamica allocata nella partita precedente.

```c
#include <stdio.h>
#include <stdlib.h> // Da includere per utilizzare rand() e srand()
#include <time.h> // Da includere per utilizzare time()

int main () {
  time_t t;

  /* Inizializza il generatore di numeri casuali utilizzando il tempo attuale */
  srand((unsigned) time(&t));  // Funzione da chiamare una volta sola nel programma

  /* Ritorna un numero tra 0 e 99 */
  printf("%d\n", rand() % 100); // Chiamare quando si ha bisogno di un numero random
}

```

Figura 1: Esempio di come effettuare la generazione di un numero casuale tra 0 e 99.

Per compilare. Includere `gamelib.h` dentro `main.c` e dentro `gamelib.c` (i.e. *#include "gamelib.h"*). A questo punto si può compilare indipendentemente `main.c` e `gamelib.c`, con rispettivamente i comandi `gcc -c main.c` e `gcc -c gamelib.c` (vengono generati rispettivamente i file `main.o` e `gamelib.o`). Infine, per comporre i due file, linkarli con `gcc -o gioco main.o gamelib.o`. Aggiungere sempre i flag *-std=c11 -Wall* (per esempio `gcc -c game.c -std=c11 -Wall`), per essere sicuri di seguire lo standard *C 2011* e farsi segnalare tutti i *warning* rispettivamente. I *warning* vanno **tutti** rimossi.

Note finali. Il testo deve essere ritenuto una traccia da seguire il più possibile, ma lo studente è libero di modificare ed estendere alcune parti, con la possibilità di ottenere punti aggiuntivi nella valutazione nei casi in cui l'estensione sia giudicata considerevole. Si possono utilizzare solamente le funzioni di libreria standard del C[2]: contattare il docente in caso si vogliano utilizzare librerie differenti. Come ispirazione, possono essere utilizzati gli esempi svolti dei progetti degli anni passati presenti sulla pagina del corso[3].

Verranno accettate solo sottomissioni attraverso il link *GitHub classroom* fornito per ogni appello (sarà sempre diverso, ricordarsi si utilizzare il link relativo all'appello orale nel quale ci si vuole presentare). Lo studente deve essere in grado di verificare da solo se il *push* delle modifiche effettuate in locale abbia avuto successo o meno (non si risponde a domande come "Può controllare se ho sottomesso correttamente?"). L'utilizzo delle funzionalità minime di GitHub fa infatti parte delle conoscenze richiesta dal corso. Per quanto riguarda il progetto di esonero, si prega di **non** creare *branch* secondari, ma di modificare il solamente template nel branch principale: al momento del download per la correzione, i branch secondari **non** sono visibili e quindi il repository risulterebbe vuoto (progetto respinto automaticamente). Infine, non è necessario avere i diritti di amministratore da parte dello studente, che infatti non sono concessi in modo da per esempio non poter rendere il repository pubblico (in quanto esercizio individuale). Nel template dell'esercizio è presente un file *.gitignore* che previene il push di file eseguibili (*.exe* o *.out*) e altri, che **non** devono far parte dal versionamento offerto da GitHub.

Ricordarsi di aggiungere nome, cognome e matricola nel file README del progetto, su GitHub. Nel file README, oppure in un altro file di testo, descrivere brevemente le ulteriori funzionalità introdotte, in caso siano state aggiunte, o comunque le scelte progettuali che si vogliono dettagliare (esempio, lettura o scrittura dati su file, etc).

[2] `https://it.wikipedia.org/wiki/Libreria_standard_del_C`.
[3] `http://www.dmi.unipg.it/francesco.santini/progI.html`.

****Inizio modifica**** Il programma verrà testato su Ubuntu 20.4, che fornisce di base un gcc versione 9.3.0. Eventuali errori, come parentesi mancanti o in più verranno considerati errori anche se un compilatore diverso da questo tollera queste mancanze. ****fine modifica****