Progetto Esame Programmazione Procedurale a.a. 2023/2024 - "Scalogna-Quest: Un gioco testuale in C"

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

5 Dicembre 2023

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. Questo gioco è liberamente ispirato a Hero Quest, un gioco da tavolo ad ambientazione fantasy, creato dalla Milton Bradley Company con la partecipazione della Games Workshop. Il gioco fu rilasciato in Europa e Australia nel 1989. Nel 2021, dopo che il marchio fu acquisito da Hasbro, quest'ultima ha lanciato un crowdfunding da un milione di euro (raggiunto in 24 ore) per il design e produzione di un remake[1]. Un gameplay in italiano può essere visto a questo link[2].

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*,

[1] `https://www.wired.it/review/ritorno-heroquest/`.
[2] `https://www.youtube.com/live/7Q_I1_tFlSk?si=C6_KMVQpG-5zT3zN`.

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 giocare 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.

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_segrete*, *enum classe_giocatore*, *enum Tipo_zona*, *enum Tipo_tesoro*, *enum Tipo_porta*:

- *struct Giocatore* contiene *i)* un array di char *nome_giocatore* di caratteri che rappresenta il nome del giocatore, *ii)* un campo *enum classe_giocatore* che descrive la classe del personaggio (vedi dopo), *iii)* un campo che memorizza la posizione sulla mappa (*struct Zona_segrete* posizione*) contenente il puntatore alla zona dove si trova il giocatore. Ogni giocatore ha *iv)* un numero di punti vita *unsigned char p_vita*, *v)* un numero di *dadi_attacco* (tipo *unsigned char*), un *vi)* numero di *dadi_difesa* (tipo *unsigned char*), una *vii)* caratteristica di *unsigned char mente*, infine *viii)* un valore di *potere_speciale* (tipo *unsigned char*).
- *struct Zona_segrete* ha due puntatori, *struct Zona_segrete* zona_successiva* e *struct Zona_segrete* zona_precedente*, per poter avanzare o tornare indietro nella mappa. Ogni zona è inoltre descritta da un *i) enum Tipo_zona*, ed altri due campi di tipo *ii) enum Tipo_tesoro* e *iii) enum Tipo_porta*.
- *enum classe_giocatore* può essere *barbaro*, *nano*, *elfo*, *mago*.
- *enum Tipo_zona* può assumere i valori *corridoio*, *scala*, *sala_banchetto*, *magazzino*, *giardino*, *posto_guardia*, *prigione*, *cucina*, *armeria*, *tempio*.
- *enum Tipo_tesoro*: *nessun_tesoro*, *veleno*, *guarigione*, *doppia_guarigione*.
- *enum Tipo_porta*: può assumere i valori *nessuna_porta*, *porta_normale*, *porta_da_scassinare*.

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). Ogni giocatore sceglie la propria classe ed in base a questo variano dei valori dei suoi campi: un barbaro ha 3 dadi attacco, 2 di difesa e 8 punti vita, il nano 2-2-7, l'elfo 2-2-6, il mago 1-2-4. Inoltre, il barbaro ha 1 o 2 di valore per la caratteristica *mente* (da decidere casualmente), il nano 2 o 3, l'elfo 3 o 4, il mago 4 o 5. Dopo questa inizializzazione, ogni giocatore può sacrificare 1 punto mente aumentando di 1 i punti vita, e viceversa. Inoltre, il valore del potere speciale è di 0 per il barbaro, 1 per nano o elfo, 3 per il mago.

  Dopodiché si deve generare la mappa di gioco, che è costituita da una lista di *struct Zona_segrete*. Per questo c'è quindi bisogno di due puntatori globali per memorizzare il primo e l'ultimo elemento della lista (*struct Zona_segrete* pFirst* e *struct Zona_segrete* pLast*). La mappa è implementata da una lista *doppiamente collegata*: ogni zona conterrà due puntatori, uno alla zona successiva, ed uno alla zona precedente. In questo modo il giocatore potrà tornare indietro sui suoi passi. 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 cinque funzioni:

1) La funzione *genera_mappa()* crea 15 zone riempiendo i campi di ciascuna con uguali probabilità, per esempio la probabilità del tipo della zona di essere un'armeria sarà di 1 su 10 (10%), quella per una porta di essere da scassinare sarà 1 su 3 (33%). Decidere se ad una chiamata successiva di questa funzione si aggiungono altre 15 zone oppure si sovrascrive le 15 zone create.
2) *inserisci_zona()* questa funzione inserisce in una posizione a piacere (in posizione *i*) una nuova zona con i campi generati casualmente.
3) *cancella_zona()* questa funzione cancella la zona in una posizione a piacere (in posizione *i*).
4) Stampa tutti i campi di tutte le zone create fino a quel momento (*stampa_mappa()*), compreso il tipo di tesoro.
5) 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()*. Non si può chiudere la mappa se ci sono meno di 15 zone nella mappa di 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. Il gioco si può impostare più volte prima di giocare.

- *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). In ogni turno di ogni giocatore, la funzione *avanza()* può essere richiamata una volta sola; decidere quando passare il turno ad un altro giocatore (per esempio con una funzione *passa()*, oppure per esempio con un numero massimo di azioni compiute nel turno corrente). Durante il turno di un giocatore, le funzioni che possono essere chiamate per giocare sono:

- *avanza()*: si avanza nella stanza successiva. Se c'è una porta (normale o da scassinare) si deve prima però aprire la porta che conduce alla stanza successiva con *apri_porta()*. Tutte le volte che si avanza in una nuova stanza, c'è una probabilità del 33% che appaia un *abitante delle segrete*. Per avanzare o indietreggiare ulteriormente, è necessario combatterlo. Nell'ultima stanza delle segrete apparirà sempre un abitante.

- *indietreggia()*: si indietreggia nella stanza precedente, dove si è già stati. In questo caso, la probabilità che appaia un *abitante delle segrete* sono del 33%. Per avanzare o indietreggiare ulteriormente, è necessario combatterlo. *Perché indietreggiare? I tesori si rigenerano, potrebbe essere conveniente per ottenere una nuova guarigione*.

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

- *stampa_zona()*: stampa i valori di (quasi) tutti i campi della zona dove si trova il giocatore. Si deve visualizzare la presenza di un tesoro o di una porta, ma non il suo tipo (per esempio se veleno o guarigione, oppure porta normale o da scassinare).

- *apri_porta()*: se la porta che si vuole aprire va scassinata, deve essere effettuato un tiro di dado (da 1 a 6): se il risultato è uguale o minore alla caratteristica *mente*, allora la porta è scassinata, ed e possibile poi avanzare nella zona successiva. Se si sbaglia, 10% si rinizia il percorso dalla prima stanza delle segrete, 50% si perde un punto vita, 40% appare un *abitante delle segrete* che si deve combattere. È possibile quindi che appaia sia perché il giocatore si è mosso in una nuova stanza, sia per colpa della porta da scassinare: ci possono essere due apparizioni nella stessa stanza.

- *prendi_tesoro()*: il giocatore prende il tesoro che c'è nella stanza in cui si trova, subendone gli effetti: con veleno perde due punti vita, con guarigione ne guadagna uno, con doppia guarigione ne guadagna due. Il tesoro si rigenera ad ogni entrata nella stanza dello stesso giocatore oppure di altri giocatori.

- *scappa()*: Invece di affrontare un abitante si può cercare di scappare: in questo caso si lancia un dado e si compare con la propria caratteristica *mente*. Se uguale o inferiore, il giocatore indietreggia nella zona precedente (in questo caso non si considera la probabilità del 33% che appaia un abitante). Altrimenti, il giocatore subisce un attacco potendosi difendere con un numero di dadi dimezzato (per difetto).

- *combatti()*: si lancia un dado da 6 per abitante e giocatore: se il risultato del giocatore è maggiore uguale di quello dell'abitante, inizia il giocatore, altrimenti l'abitante. I dadi da combattimento (sono gli stessi per attacco e difesa) sono dadi a 6 facce con pittogrammi al posto dei numeri. Giocatore e abitante lanciano tutti i loro dadi di attacco per attaccare, e tutti i loro dadi difesa per difendersi. Ogni dado ha 3 teschi, 2 scudi bianchi e 1 scudo nero sui lati. Ogni teschio lanciato è considerato un colpo, messo a segno contro il giocatore/mostro, e sottrae un punto vita se l'attacco non viene difeso. ll mancato lancio di un teschio risulta in un attacco fallito. Se colpito, il giocatore/abitante si difende immediatamente lanciando tutti i suoi dadi di difesa. Il giocatore para un attacco per ogni scudo bianco che ottiene, mentre l'abitante per ogni scudo nero.[3] Se i punti vita di giocatore o abitante raggiungono lo zero, la vittima viene rimossa dal gioco. Chi si è difeso inizialmente, poi lancia i suoi dadi attacco e si ripete la procedura di difesa. A questo punto è terminato il primo turno e si rilancia il dado da 6 per capire chi attaccherà per primo nel secondo turno, e così via fino alla morte di uno dei due duellanti.

- *gioca_potere_speciale()*: si può giocare potere speciale invece di combattere, per uccidere immediatamente l'abitante che il giocatore si trova davanti. Il numero di questo potere decrementa di uno ogni volta che viene utilizzato.

Abitante delle segrete: Le caratteristiche dell'abitante devono essere generate casualmente. Le caratteristiche sono costituite da un nome, un numero di dadi attacco, un numero di dadi difesa, un numero di punti vita. Lo studente definisca in completa autonomia come generare queste caratteristiche tra un minimo ed un massimo di valori; per esempio in proporzione al numero di turni passati potrebbero aumentare le caratteristiche, oppure totalmente a caso ed uguali per ogni turno.

Vince il giocatore che arriva in fondo al percorso per primo. 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.

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`).

[3] `https://heroquest.fandom.com/wiki/Guide:Attacking_and_defending`.

```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 generare un valore tra 1 e 6 le considerazioni sono simili, basta cambiare il modulo.

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[4]: 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[5].

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).

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.

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