Progetto Esame Programmazione Procedurale a.a. 2024/2025 - "Prince of Inertia"

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

2 Dicembre 2024

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. Il gioco è ambientato nella Persia medioevale. Mentre il Sultano è in guerra lontano dal paese, il malvagio Visir Jaffar complotta per salire al trono. L'ultimo ostacolo che lo separa dal potere è costituito dalla Principessa, l'unica figlia del sovrano, cui ha concesso un'ora per decidere tra il matrimonio con lui o la morte. Nel frattempo Jaffar ha fatto imprigionare il giovane amato dalla Principessa, che viene confinato nelle segrete del palazzo del Sultano. Il giovane riesce a recuperare una spada appartenuta a un guerriero ormai ridotto a un cumulo di ossa, e tenta di fuggire fronteggiando le guardie armate del Visir. Chi riuscirà a raggiungere per primo la principessa? Il Principe o i suoi Doppelgänger. Il gioco è liberamente ispirato a Prince of Persia.[1] È possibile trovare il completo walkthrough su Youtube.[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, *4)* visualizza i crediti. *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

[1] Prince of Persia è un videogioco a piattaforme sviluppato da Jordan Mechner e pubblicato da Brøderbund nel 1989, originariamente per Apple II: `https://it.wikipedia.org/wiki/Prince_of_Persia`.

[2] `https://www.youtube.com/watch?v=FGQmtlxllWY`.

per effettuare la scelta richiesta. In caso il comando non sia 1-2-3-4, 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, 3 o 4, chiamare la corrispondente funzione definita in *gamelib.c*, cioè *imposta_gioco()*, *gioca()*, *termina_gioco()*, o *crediti()*. 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 Stanza*, *enum classe_giocatore*, *enum Tipo_stanza*, *enum Tipo_tesoro*:

- *struct Giocatore* contiene come campi per lo meno *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 Stanza* posizione*) contenente il puntatore alla stanza dove si trova il giocatore. Ogni giocatore ha *iv)* un numero di punti vita massimo *unsigned char p_vita_max*, *v)* un numero di punti vita corrente *unsigned char p_vita vi)* un numero di *dadi_attacco* (tipo *unsigned char*), un *vi)* numero di *dadi_difesa* (tipo *unsigned char*).
- *struct Stanza* contiene come campi per lo meno quattro puntatori, *struct Stanza* stanza_destra*, *struct Stanza* stanza_sinistra*, *struct Stanza* stanza_sopra*, *struct Stanza* stanza_sotto* per poter avanzare in ogni direzione della mappa. Ogni stanza è inoltre descritta da un campo di tipo *enum Tipo_stanza*, un campo *enum Tipo_trabocchetto*, ed un campo *enum Tipo_tesoro*.
- *enum tipo_giocatore* può essere *principe* o *dopplegänger*.
- *enum Tipo_stanza* può assumere i valori *corridoio*, *scala*, *sala_banchetto*, *magazzino*, *posto_guardia*, *prigione*, *armeria*, *moschea*, *torre*, *bagni*.
- *enum Tipo_trabocchetto*: può assumere i valori *nessuno*, *tegola*, *lame*, *caduta*, *burrone*.
- *enum Tipo_tesoro*: *nessun_tesoro*, *verde_veleno*, *blu_guarigione*, *rosso_aumenta_vita*, *spada_tagliente*, *scudo*.

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 3. Ogni giocatore è rappresentato da una *struct Giocatore*, che viene creta in memoria dinamica. L'insieme dei giocatori è rappresentato da un array di 3 *Struct Giocatore** (*NULL* se il giocatore non partecipa al gioco). Se un giocatore sceglie *principe*, i successivi potranno essere solo *dopplegänger*. Ci deve essere un *principe* per poter giocare. Tutti i giocatori iniziano con i valori di *p_vita* e *p_vita_max* pari a 3 (modificare a piacimento per bilanciare il gioco).

  Dopodiché si deve generare la mappa di gioco, che è costituita da una lista di *struct Stanza*. Per questo c'è quindi bisogno di un puntatore globale per memorizzare il primo o elemento della lista (*struct Stanza* pFirst*). 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:

  La funzione *imposta_gioco()* viene richiamata da *main.c* e serve per richiedere al creatore della mappa le operazioni da effettuare sulla mappa; anche in questo caso, stampare un menu e leggere la risposta da tastiera. Esse corrispondono alle funzioni *ins_stanza()*, *canc_stanza()*, *stampa_stanze()*, *genera_random()* e *chiudi_mappa()*.

Tutte le seguenti funzioni devono essere definite come *static*, in quanto non devono essere chiamate da fuori *gamelib.c*:

La funzione *imposta_gioco()* può essere chiamata più volte in sequenza , se per esempio si vuole reimpostare il gioco.

1) *ins_stanza()*: inserimento di una stanza in coda rispetto all'ultima caverna già creata. Essa crea la nuova caverna in memoria dinamica (*malloc()*), la inserisce nella lista modificando il valore del puntatore, a seconda che il creatore scelga di inserire la caverna sopra, a destra, a sinistra, sotto. Posso creare la prossima caverna soltanto lungo una delle quattro direzioni. La prima inserzione, quando la lista è vuota, non distingue questa direzione (in pratica, esiste un solo puntatore ad inizio lista). Tutti i valori della nuova *struct Stanza* devono essere letti da tastiera.
2) *canc_stanza()*: cancella l'ultima stanza inserita nella mappa. Essa libera la memoria occupata dall'ultima caverna (*free()*).
3) *stampa_stanze()*: stampa i campi di tutte le stanze create fino a quel momento, compresi tutti i valori dei puntatori che sono campi di una stanza (in esadecimale).
4) Con *genera_random()*, tutte le stanze create precedentemente (se ce ne sono) vengono cancellate, e viene creata una nuova mappa di 15 stanze i cui valori dei campi vengono scelti in modo casuale. Da notare che si può continuare ad inserire (con *ins_stanza()*) e cancellare (con *canc_stanza()*) stanze a partire dalle 15 create con questa funzione. Questa funzione non deve essere chiamata di default per creare una mappa iniziale: è a scelta del game master chiamarla oppure no. Ciò vuol dire, per esempio, che si può scegliere di generarla randomicamente e poi modificarla con le altre funzioni di inserzione e cancellazione, oppure si può scegliere di generarla subito con queste funzioni. Come detto, per ogni stanza i valori dei campi si generano casualmente nella seguente maniera: il tipo della stanza è equiprobabile (10% per ciascuno dei 10 valori), per il 65% non esiste alcun trabocchetto nella stanza, mentre i 4 tipi di trabocchetto hanno rispettivamente *tegola* 10%, *lame* 9%, *caduta* 8%, *burrone* 8%. Infine per il tipo del tesoro, rispettivamente *nessun_tesoro* 20%, *verde_veleno* 20%, *blu_guarigione* 20%, *rosso_aumenta_vita* 15%, *spada_tagliente* 15%, *scudo* 10%.
5) Fine della creazione della mappa con *chiudi_mappa()*. Ci si ricorda che la mappa è stato terminata (per esempio settando una variabile globale e *static* da 1 a 0), e si esce anche dalla funzione *imposta_gioco()*. Non si può chiudere la mappa anche se ci sono meno di 15 stanze inserite nella mappa.

- *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. Da ricordarsi che il gioco si può impostare più volte prima di giocare. Anche in questo caso le funzioni devono essere definite come *static*.

  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 2-3-1, e poi si riparte, sempre con un nuovo ordine casuale. In ogni turno di ogni giocatore, la funzione *avanza()* può essere richiamata una volta sola; decidere quando passare il turno ad un altro giocatore con la funzione *passa()*. Durante il turno di un giocatore, le funzioni che possono essere chiamate per giocare sono:

  1) *avanza()*: si avanza nell'unica direzione possibile (sopra, destra, sinistra, sotto) che viene stampata sullo schermo. L'effetto del trabocchetto presente nella stanza viene attivato. Per ogni volta che si

avanza, c'è una probabilità del 25% che appaia un nemico (60% uno scheletro, 40% una guardia). Nell'ultima stanza si trova sempre Jaffar.

2) *combatti()*: ogni giocatore inizia con 2 dadi attacco e 2 dadi difesa; tutti i dadi si considerano da 6 facce. Gli scheletri hanno 1 dado attacco - 1 dado difesa - 1 punto vita, le guardie hanno 2 dadi attacco - 2 dadi difesa - 2 punti vita, Jaffar ha 3 dadi attacco - 2 dadi difesa - 3 punti vita. Se uno di questi nemici è apparso nella stanza, si può decidere di combattere utilizzando questa funzione. I due combattenti tirano 1 dado a 6 facce, chi ottiene il punteggio più alto attacca per primo e l'altro si difende, poi attacca l'altro ed il primo si difende. Ad ogni turno di combattimento si ritira il dado per sapere chi attacca per primo. Ogni dado attacco che ottiene 4 o più è considerato un colpo riuscito. Ogni dado che ottiene 4 o più neutralizza un colpo riuscito dell'avversario. Gli attacchi non neutralizzati, provocano 1 danno. Critico d'attacco: un 6 naturale su un dado attacco infligge 2 danni invece di 1 (se non neutralizzato). Critico di difesa: un 6 naturale su un dado difesa neutralizza due colpi invece che uno. Per esempio, se l'attaccante totalizza 5 e 4, e il difensore 2 e 5, il difensore subirà 1 danno; se l'attaccante totalizza 6 e 6, e il difensore 2 e 6, il difensore avrà parato tutti e due gli attacchi subendo così 0 danni; se l'attaccante totalizza 3 e 6, e il difensore 3 e 1, il difensore subirà 2 danni derivanti dal colpo critico. Per ogni combattimento vinto si guadagna 1 punto vita. Per ile combinazioni particolari, si lascia libertà di scelta progettuale.

3) *scappa*: in caso sia comparso un nemico, si può decidere di scappare. Un principe può scappare solo una volta, mentre un dopplegänger due volte. Di contro, un principe ignora gli effetti del primo trabocchetto in cui cade vittima. Chi scappa ritorna nella stanza da cui era avanzato. Ogni giocatore può subire gli effetti di un trabocchetto una volta sola, in caso si riposti in una stana dove era già stato.

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

5) *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, ma non il suo tipo (per esempio se veleno).

6) *prendi_tesoro()*: il giocatore prende il tesoro che c'è nella stanza in cui si trova (se c'è), subendone gli effetti (vedere dopo). Una volta consumato preso da un giocatore, il tesoro non esiste più per quella stanza.

7) *cerca_porta_segreta*: in ciascuna stanza, per ognuna delle tre direzioni in cui non è possibile procedere, è possibile cercare una sola volta una stanza segreta. La prima volta che si cerca la probabilità di trovare una stanza segreta è del 33%, la seconda 20%, la terza 15%. Se trovata, la nuova stanza viene aggiunta alla mappa (*malloc*), ed il giocatore si sposta automaticamente in quella stanza. I valori di questa stanza vengono generati casualmente come nella funzione *genera_random()*. In queste stanza segrete non possono apparire nemici. Infine, il giocatore ritorna automaticamente alla stanza in cui aveva cercato la porta segreta. Se un giocatore avanza in un turno, non può cercare porte segrete, e viceversa.

8) *passa()*: si passa il turno al prossimo giocatore.

- *termina_gioco()* si termina il gioco salutando i giocatori.
- *crediti()* si mostra il nome del creatore del gioco ed i nomi dei vincitori delle ultime tre partite (i nomi dei giocatori).

Se il giocatore raggiunge gli zero punti vita, muore. Il gioco termina quando tutti i giocatori hanno raggiunto zero punti vita, o quando il primo dei giocatori sconfigge Jaffar, risultando come vincitore della partita. Quando il gioco termina, è possibile reimpostare il gioco (*imposta_gioco()*) e giocare nuovamente su una nuova mappa.

```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.

Ricordarsi di deallocare (con *free*) tutta la memoria dinamica allocata giocando precedentemente (per esempio, la lista delle stanze).

Di seguito si riporta gli effetti dei tesori e dei trabocchetti.

- *tegola* si perde un punto vita.
- *lame* si perdono due punti vita
- *caduta* si perdono 1 o 2 punti vita, con eguale probabilità.
- *burrone* come caduta, ma si perde anche un turno (cioè negli altri turni giocheranno solo gli altri giocatori).
- *verde_veleno*: viene diminuito di uno punto *p_vita* del giocatore.
- *blu_guarigione*: viene aumentato di uno punto *p_vita* del giocatore, fino al massimo di *p_vita_max*.
- *rosso_aumenta_vita*: viene aumentato di uno il valore massimo di punti vita (*p_vita_max*), riportando anche al massimo *p_vita*.
- *spada_tagliente* viene aumentato di uno il numero di dadi di attacco del giocatore (*dadi_attacco*).
- *scudo*: viene incrementato di uno il numero di dadi difesa del giocatore (*dadi_difesa*).

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[3]: 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[4].

Verranno accettate solo sottomissioni attraverso il link *GitHub classroom* fornito per ogni appello che sarà pubblicizzato sia sulla pagina Web del corso, sis su Unistudium. Il link sarà sempre diverso, ricordarsi si utilizzare il link relativo all'appello orale nel quale ci si vuole presentare. Attenzione: risottomettere sempre il progetto con il link dell'appello a cui si ci vuole presentare, anche se il progetto è giò stato sottomesso a link di appelli precedenti. 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. 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 24.04.1, che fornisce di base almeno un gcc versione 13.2.0. Eventuali errori, come parentesi mancanti o in più verranno considerati errori anche se un compilatore diverso da questo tollera queste mancanze.

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