Progetto Esame Programmazione Procedurale a.a. 2025/2026 - "Cosestrane"

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

9 Dicembre 2025

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 dei tipi delle strutture dati utilizzate in `gamelib.c`.

La storia. Nella tranquilla cittadina di Occhinz, famosa per i Waffle Undici e per il numero inspiegabilmente alto di biciclette scomparse, cominciano ad aprirsi strani portali dimensionali, collegando il Mondo Reale a una versione oscura, polverosa, appiccicosa e decisamente poco accogliente: il temuto Soprasotto. I giocatori si ritroveranno a esplorare entrambi i mondi, muovendosi tra luoghi familiari come boschi, strade e strutture abbandonate, e le loro versioni distorte. Qui dovrà evitare minacce come il Demotorzone e altre presenze che infestano la dimensione oscura, cercando nel frattempo di trovare un'uscita verso la realtà.

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 per effettuare la scelta richiesta. In caso in cui il comando non sia 1-2-3-4, stampare al giocatore un messaggio che il comando è sbagliato e poi ristampare il menu. Eseguire controlli simili sull'input anche per tutte le altre letture da tastiera presenti 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 (liberando ogni volta tutta la memoria dinamica allocata in precedenza). 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 quattro funzioni introdotte precedentemente, e le definizioni dei tipi utilizzati nella libreria, cioè i tipi *struct Giocatore*, *struct Zona_mondoreale*, *struct Zona_soprasotto*, *enum Tipo_zona*, *enum Tipo_nemico* e *enum Tipo_oggetto*:

- La *struct Giocatore* rappresenta ciascun giocatore. Il campo *nome* memorizza il nome scelto dall'utente (una stringa di caratteri). Il campo *mondo* specifica in quale delle due dimensioni si trova attualmente il personaggio, con 0 per il Mondo Reale e 1 per il Soprasotto. Poiché il giocatore può trovarsi soltanto in una delle due mappe alla volta, la struttura contiene due puntatori separati: *pos_mondoreale*, che indica la zona del Mondo Reale in cui si trova se il campo mondo vale 0, e *pos_soprasotto*, che punta alla zona corrispondente nel Sottosopra quando il campo *mondo* vale 1. Ciascun giocatore ha anche tre abilità (ossia, capi della struttura): *attacco_pischico*, *difesa_pischica*, *fortuna*, che assumono un valore da 1 a 20. Infine, il giocatore si può portare dietro fino ad un massimo di 3 oggetti di tipo *enum Tipo_oggetto*: rappresentare il campo *zaino* come un array.

- *struct Zona_mondoreale* rappresenta una singola area della mappa del Mondo Reale e costituisce un nodo di una lista doppiamente collegata che modella la città di Occhinz e i suoi dintorni. Ogni zona è descritta da un campo *tipo* (di tipo *enum Tipo_zona*). Inoltre, sono presenti i campi *nemico* di tipo *enum Tipo_nemico* e *oggetto* di tipo *enum Tipo_oggetto*. I campi *avanti* e *indietro* sono puntatori che collegano la zona rispettivamente alla zona successiva e a quella precedente della mappa, consentendo di percorrerla in entrambe le direzioni. Il campo *link_soprasotto* è un puntatore diretto alla zona corrispondente nell'altra mappa, quella del Soprasotto.

- *struct Zona_soprasotto* rappresenta una singola area della mappa del Soprasotto e costituisce anch'essa un nodo di una lista doppiamente collegata che modella la versione speculare di Occhinz. Ogni zona è descritta da un campo *tipo* (di tipo *enum Tipo_zona*). Anche in questo caso è presente il campo *nemico* di tipo *enum Tipo_nemico*, ma NON è presente il campo *oggetto*. I campi *avanti* e *indietro* sono puntatori che svolgono anche qui la stessa funzione di potersi muovere avanti e indietro in questa lista. Il campo *link_mondoreale* è un puntatore diretto alla zona corrispondente nell'altra mappa, quella del Soprasotto.

- *enum Tipo_zona* può assumere i valori *bosco*, *scuola*, *laboratorio*, *caverna*, *strada*, *giardino*, *supermercato*, *centrale_elettrica*, *deposito_abbandonato*, *stazione_polizia*.

- *enum Tipo_nemico*: può assumere i valori *nessun_nemico*, *billi*, *democane*, *demotorzone*.

- *enum Tipo_oggetto*: può assumere i valori *nessun_oggetto*, *bicicletta*, *maglietta_fuocoinferno*, *bussola*, *schitarrata_metallica*.

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()*, *termina_gioco()*, o *crediti()* devono essere definite con lo specificatore *static*, dato che non devono essere visibili all'esterno di questo file.

Possono essere aggiunte funzioni a piacere per personalizzare il gioco rispetto a questa traccia (indicare le modifiche/aggiunte su GitHub, come descritto al termine di questo documento).

La funzione *imposta_gioco()* può essere chiamata più volte in sequenza per reimpostare il gioco dopo averlo già impostato.

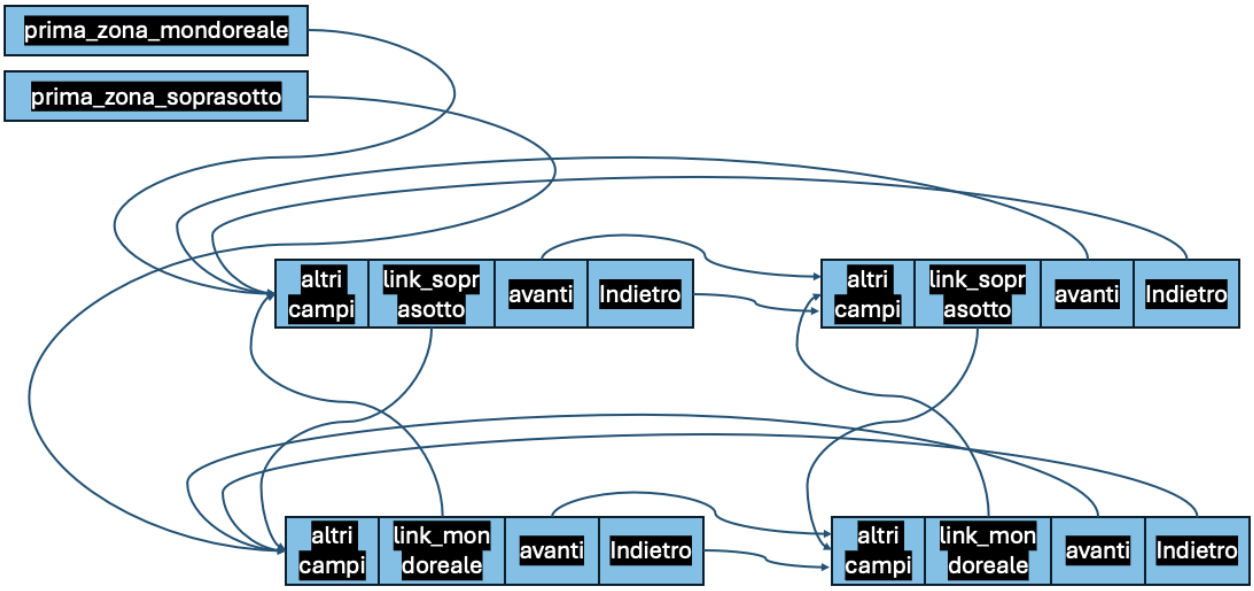

Figura 1: Un esempio di mappa di gioco con due sole zone.

- *imposta_gioco()*. Questa funzione chiede, come prima cosa, di inserire da tastiera il numero di giocatori, che può variare tra 1 e 4. Ogni giocatore è rappresentato da una *struct Giocatore* allocata in memoria dinamica. L'insieme dei giocatori è rappresentato da un array di 4 *Struct Giocatore** (*NULL* se il giocatore non partecipa al gioco o muore). A ogni giocatore si chiede di inserire il proprio *nome*, e poi si lancia un dado da 20 per stabilire i valori di *attacco_pischico*, *difesa_pischica* e *fortuna*. Ogni giocatore può a questo punto scegliere se aumentare *attacco_pischico* di 3 punti diminuendo *difesa_pischica* di 3 punti, o viceversa. Un giocatore può anche scegliere di diventare *UndiciVirgolaCinque* aumentando *attacco_pischico* e *difesa_pischica* di 4 punti, ma diminuendo la *fortuna* di 7 punti. Un solo giocatore per partita può diventare *UndiciVirgolaCinque*, ed il suo nome cambia di conseguenza. All'inizio del gioco, lo zaino è vuoto per ciascun giocatore.

  Dopodiché, si deve generare la mappa di gioco, costituita da due liste, una di *struct Zona_mondoreale* e una di *struct Zona_soprasotto*. Per questo c'è quindi bisogno di un puntatore globale per memorizzare la prima della mappa che rappresenta il Mondo Reale (*struct Zona_mondoreale* prima_zona_mondoreale*), e analogo puntatore per la lista che rappresenta la mappa del Soprasotto (*struct Zona_soprasotto* prima_zona_soprasotto*). Le due mappe sono implementate come una lista *doppiamente collegata*. In questo modo, il giocatore potrà tornare sui propri passi. Un esempio di mappa con due sole zone è mostrato in Figura 1. 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, assegnando casualmente a ciascuna il suo *tipo* con probabilità uguali: per esempio, una zona è una scuola con probabilità 1 su 10 (10%). Generare anche il *nemico* e l'*oggetto* (quest'ultimo presente solo se la zona è del Mondo Reale), con probabilità tali da garantire un minimo di giocabilità. *nessun_nemico* e *democane* possono essere presenti in entrambi i mondi, mentre *billi* è presente solo nel Mondo Reale e *demotorzone* solo nel Soprasotto. Un solo *demotorzone* deve essere SEMPRE presente nella mappa del Soprasotto essendo condizione per la vittoria dei giocatori (vedere in seguito). Ad una chiamata successiva di questa funzione si sovrascriveranno le 15 zone create con altre 15, eliminando tutte quelle precedenti. In questo modo saranno create 15 zone per il Mondo Reale e 15 corrispondenti per il Soprasotto. Il tipo della zona

corrispondente del Soprasotto è lo stesso di quello del Mondo Reale, mentre il campo *nemico* può variare.

2) *inserisci_zona()* questa funzione inserisce in una posizione a piacere (in posizione *i*) una nuova zona con i campi generati casualmente (come in *genera_mappa()*). Allo stesso modo, si inserisce la zona speculare corrispondente nella lista delle zone del Soprasotto. Chiedere all'utente (cioè far scegliere) i valori dei campi *nemico* e *oggetto*, inserendoli dalla tastiera e rispettando gli stessi vincoli della funzione *genera_mappa()*).

3) *cancella_zona()* questa funzione cancella la zona in una posizione a piacere (in posizione *i*). Allo stesso modo, si cancella dalla lista delle zone del Soprasotto la zona speculare corrispondente.

4) Stampa tutti i campi di tutte le zone create fino a quel momento (*stampa_mappa()*). A scelta dell'utente, questa funzione stampa tutta la mappa del Mondo Reale o quella del Soprasotto.

5) La funzione *stampa_zona()*) stampa tutti i campi di una delle zone del Mondo Reale a scelta dell'utente (in posizione *i*) e di quella collegata ad essa nel Soprasotto.

6) Fine della creazione della mappa: *chiudi_mappa()*. Ci si ricorda che la creazione della mappa è terminata (per esempio, impostando una variabile globale e statica da 0 a 1). Questa variabile viene controllata quando si chiama *gioca()*, per verificare 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, oppure se non è presente uno (e uno solo) *demotorzone*.

- *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 nella prima zona della mappa del Mondo Reale. Anche in questo caso, le funzioni devono essere definite come *static* perché non sono visibili all'esterno della libreria di gioco.

  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 giocatori rimanenti devono aver giocato il loro; ad esempio, tre 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 sola volta; per passare il turno a un altro giocatore, usare la funzione *passa()*. Durante il turno di un giocatore, le funzioni che possono essere chiamate per giocare sono:

  1) *avanza()*: si avanza nella zona successiva a quella in cui ci si trova (rispetto alla mappa in cui ci si trova, o del Mondo Reale o del Soprasotto). Prima di avanzare, si deve combattere il *nemico*, se nella stanza in cui ci si trova ce n'è uno.

  2) *indietreggia()*: si indietreggia nella zona precedente a quella in cui ci si trova (rispetto alla mappa del Mondo Reale o Soprasotto in cui ci si trova). Prima di indietreggiare, si deve combattere il *nemico*, se nella stanza in cui ci si trova ce n'è uno.

  3) *cambia_mondo()*: se ci si trova nella mappa del Mondo Reale, con questa funzione si viene catapultati in quello del Soprasotto, cioè nella zona del Soprasotto corrispondente a quella in cui si trova il giocatore; se ci si trova nel Soprasotto, con questa funzione si torna al Mondo Reale (sempre nella zona corrispondente a quella in cui ci si trova). Vale come una *avanza()* in questo caso e non si può quindi chiamare *cambia_mondo()* se *avanza()* è già stata chiamata in questo turno del giocatore, oppure se non si è sconfitto il nemico del Mondo Reale. Nel caso in cui il giocatore si trovi sulla mappa del Soprasotto, questa funzione si può chiamare senza vincoli, quindi anche per scappare da un eventuale *nemico*; è necessario, però, prima lanciare un dado da 20 e ottenere un punteggio inferiore alla propria *fortuna*.

```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 2: Esempio di come generare un numero casuale tra 0 e 99. Per generare un valore tra 1 e 6 le considerazioni sono simili, basta cambiare il modulo.

4) *combatti()*: se ci si trova in una stanza con un *nemico*, questa funzione avvierà un sottomenu di combattimento. Implementare la propria procedura di combattimento utilizzando almeno le caratteristiche *attacco_pischico*, *difesa_pischica* e *fortuna*, ed eventualmente anche gli oggetti posseduti dal giocatore nel suo *zaino* (vedi sotto, nella funzione *utilizza_oggetto()*). Il giocatore può morire durante il combattimento. I diversi nemici possono avere caratteristiche differenti. Dopo che un giocatore ha sconfitto un nemico, c'è un 50% di probabilità che questi scompaia dalla zona, quindi i giocatori che entrano in quella zona in seguito non siano costretti a combatterlo.
5) *stampa_giocatore()*: stampa i valori di tutti i campi del giocatore.
6) *stampa_zona()*: stampa i valori di tutti i campi della zona in cui si trova il giocatore.
7) *raccogli_oggetto()*: in una zona con un oggetto è possibile raccoglierlo, purché ci sia spazio nello zaino. Non si può raccogliere un oggetto se prima non si è ucciso il nemico, se è presente nella stanza in cui ci si trova.
8) *utilizza_oggetto()*: l'oggetto viene utilizzato. Per ciascuno dei possibili oggetti (vedi *enum Tipo_oggetto*) implementare un effetto diverso. Un oggetto può essere utilizzato anche durante la fase di combattimento, se si desidera.
9) *passa()*: il giocatore cede il proprio turno al prossimo giocatore.

- *termina_gioco()* termina il gioco salutando i giocatori.
- *crediti()* mostra il nome del creatore del gioco, i nomi dei vincitori delle ultime tre partite (i giocatori) ed eventuali altre statistiche della partita.

Il primo giocatore a sconfiggere il *demotorzone* è il vincitore. Il gioco termina anche in caso in cui tutti i giocatori muoiano. Quando il gioco termina, è possibile reimpostare il gioco (*imposta_gioco()*) e giocare nuovamente su una nuova mappa. Ricordarsi di deallocare (con *free*) tutta la memoria dinamica allocata giocando precedentemente (per esempio, la lista delle stanze). Ricordarsi di deallocare anche i giocatori durante il gioco, qualora muoiano.

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 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*. 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[1]: contattare il docente in caso si vogliano utilizzare librerie differenti. Come ispirazione, possono essere utilizzati gli esempi svolti nei progetti degli anni passati presenti sulla pagina del corso[2].

Verranno accettate solo le sottomissioni tramite il link *GitHub Classroom* fornito per ogni appello, che sarà pubblicizzato sia sulla pagina Web del corso sia 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 vuole presentare, anche se il progetto è già stato sottomesso con il link di appelli precedenti. Lo studente deve essere in grado di verificare da solo se il *push* delle modifiche effettuate in locale è andato a buon fine (non si risponde a domande come "Può controllare se ho sottomesso correttamente?"). L'utilizzo delle funzionalità minime di GitHub rientra infatti tra le conoscenze richieste 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 pubblico il repository (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 del versionamento offerto da GitHub.

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

Il programma verrà testato su Ubuntu 24.04.3 LTS, che fornisce di base almeno la versione 13.2 di GCC. Eventuali errori, come parentesi mancanti, verranno inoltre considerati errori, anche se un compilatore diverso da questo le tollera.

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