Progetto Esame Programmazione Procedurale a.a. 2021/2022 - "Tragic: Un gioco di carte"

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

7 Dicembre 2021

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. Due maghi si affrontano in un duello uno-contro-uno su una piana ventosa vicina ad Ravnika, la Città delle Gilde. Gli incantesimi che possono lanciare sono rappresentati da un mazzo di carte ciascuno.[1,2]

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)* combatti, *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()*, *combatti()*, 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

[1] `https://youtu.be/4mhEjn03SLg?t=63`.
[2] `https://www.youtube.com/watch?v=TJeoqnWcmAs`.

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.

gamelib.h. Questo file deve contenere solamente le dichiarazioni delle funzioni presentate precedentemente, e le definizioni dei tipi utilizzati nella libreria, cioè i tipi *struct Mago* e *struct Carta*, il tipo *enum Classe_mago*, ed il tipo *enum Tipo_carta*:

- *struct Mago*, contiene *i)* un campo che rappresenta il nome del giocatore, *ii)* il numero di Punti Vita (PV) del mago, inizializzato a 20, *iii)* la classe del Mago (di tipo *enum Classe_mago*). Ogni struct ha anche i tre campi *mazzo*, *mano* e *campo* (ognuno è un *struct Carta**), spiegati nel seguito di questo testo.
- *enum Classe_mago* deve rappresentare i valori {*tenebre, vita, luce*}. Queste classi influiscono sull'effetto delle carte giocate.
- *struct Carta* contiene *i)* il tipo della carta, di tipo *enum Tipo_carta*, *ii)* un campo *punti_vita* che rappresenta il numero di punti vita della creatura (se la carta ha tipo *creatura*) o alternativamente il numero di punti vita inflitti (se il tipo della carta è *infliggi_danno* o *cura_danno*), *iii)* un puntatore alla successiva carta nel mazzo in cui si trova.
- *enum Tipo_carta* deve rappresentare i valori che può assumere il tipo della carta e cioè {*creatura, rimuovi_creatura, infliggi_danno, guarisci_danno*}.

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()*, *combatti()*, e *termina_gioco()* devono essere definite come *static*, dato che non devono essere visibili all'esterno di questo file.

Strutture dati di gioco Per ciascuno dei due giocatori si definiscano per lo meno le tre seguenti strutture dati, il puntatore alle quali si trova nella *struct Mago* corrispondente al giocatore:

- Un *mazzo* per ogni giocatore, rappresentato da una lista dinamica di *struct Carta* in cui si inserisce in coda (quando si crea il mazzo per ciascun giocatore) e si cancella in coda (quando si pesca una carta dal mazzo). Il puntatore alla lista di carte (inizio mazzo) si trova nella *struct Carta* del mago corrispondente.
- Un array *mano* di 6 puntatori a *struct Carta* che rappresenta la mano corrente di ciascun giocatore,che quindi può avere al massimo 6 carte in mano. Le carte pescate (o meglio il loro puntatore) vengono inserite in questo array, e vengono spostate in *campo* quando vengono giocate. Gli slot non occupati da una carta valgono quindi NULL. Questo array, o meglio il suo puntatore, si trova nella *struct Carta* del mago corrispondente.
- Un array *campo* di 4 puntatori a *struct Carta* che rappresenta le carte giocate da ciascun giocatore, che quindi ad un dato istante possono essere non più di 4. Anche in questo caso, gli slot non occupati da una carta valgono quindi NULL. Questo array, o meglio il suo puntatore, si trova nella *struct Carta* del mago corrispondente.

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

- La funzione *imposta_gioco()* deve creare le due *struct Mago*, una per ogni giocatore. Deve quindi richiedere a ciascuno dei due giocatori di inserire il nome e la classe del mago, e poi generare due mazzi di n carte (*mazzo*), dove n è letto da tastiera, e riempire le due mani (*mano*) con 5 carte in più: essa

rappresenta la mano iniziale di ciascun giocatore. Il campo di gioco sarà vuoto (tutti i puntatori di *campo* saranno inizializzati a NULL). I dati delle carte sono generati casualmente: per esempio, i quattro tipo possibili hanno rispettivamente probabilità (40%, 25%, 20%, 15%), mentre il campo *punti_vita* di ciascuna carta deve essere casuale, ma deve diminuire gradualmente con l'aumentare delle carte nel mazzo (per esempio, le prime creature in cima al mazzo devono avere 1-2 punti vita, l'ultima in fondo al mazzo 8). In questo modo si simula il fatto che le carte più potenti si possano lanciare solo con il passare dei turni. Implementare a piacere e commentare nel README di Github la scelta effettuata. A seconda della classe del mago, la generazione casuale delle carte può subire delle modifiche: se il mago ha classe *vita*, le sue carte *creatura* hanno la metà dei *punti_vita* in più, se *tenebra* le sue carte *infliggi_danno* hanno il doppio dei *punti_vita*, se *luce* le sue carte *guarisci_danno* hanno tre volte e mezzo il valore dei *punti_vita*. Come risultato finale di *imposta_gioco()*, si stampi a video tutti questi dati per entrambi i giocatori (stampare ciascuna carta su una sola linea di testo, in modo da rendere più compatto l'output). *imposta_gioco()* può essere chiamata anche più volte di fila se si desidera reimpostare il gioco; deallocare tutta la memoria dinamica creata prima di ricrearne di nuova.

- La funzione *combatti()* sceglie a caso chi dei due giocatori gioca per primo, e passa alle funzioni di gioco (vedere in seguito). In seguito ad ogni turno si alterneranno i due giocatori.
- La funzione *termina_gioco()* saluta l'utente e termina il gioco.

Una volta iniziato il gioco con *combatti()*, ad ogni turno un giocatore può pescare, giocare una carta, ed attaccare una volta: massimo una azione per ciascuna di queste possibilità, quindi nel proprio turno si può sia pescare, che giocare, che attaccare (o nessuna di queste). Al contrario si può stampare la propria *mano* ed il *campo* di entrambi i giocatori tutte le volte che desidera:

- *pesca()*: questa funzione estrae dalla cima del mazzo una carta e la sposta nella *mano* (se c'è posto, altrimenti si dealloca direttamente).
- *stampa_mano()*: stampa tutte le carte della propria mano.
- *stampa_campo()*: stampa tutte le carte della presenti sul campo.
- *gioca_carta()*: in questo caso il giocatore sceglie una delle proprie carte e sceglie un bersaglio su cui giocarla: la carta di tipo *rimuovi_creatura* può avere come bersaglio una delle creature del nemico sul *campo*, mentre *infliggi_danno* può essere giocata contro una creatura del nemico sul *campo* o direttamente contro il mago nemico , *guarisci_danno* può essere giocata a favore di sé stessi oppure su una delle proprie creature. Una *creatura* può essere invece giocata inserendola sul proprio *campo*. Una carta *infliggi_danno* infligge una danno pari al numeri dei suoi *punti_vita* ad una creatura o al mago nemico, la *guarisci_danno* incrementa i PV del mago o i *punti_vita* della creatura bersaglio, la *rimuovi_creatura* distrugge la creatura in campo (*free*). Le carte giocate (ad eccezione delle creature giocate) possono essere subito dopo deallocate (*free*). Ovviamente anche le creature, quando raggiungono 0 *punti_vita*, sono eliminate dal *campo* (*free*).
- *attacca()*: per ciascuna delle creature sul proprio campo, il giocatore può scegliere se attaccare una qualsiasi delle creature sul campo nemico, oppure il mago nemico. In caso attacchi il mago nemico, si sottraggono dal suo numero di PV i *punti_vita* correnti della creatura. In caso attacchi un'altra creatura, ad ognuna si sottrae l'ammontare corrente dei *punti_vita* dell'altra creatura. Se ina creatura raggiunge 0 *punti_vita*, essa è eliminate dal *campo* (*free*).
- *passa()*: finisce il turno di un giocatore, passando all'altro la possibilità di giocare.

Il gioco termina quando uno dei due giocatori arriva a 0 PV, oppure, quando il primo giocatore termina le carte nel mazzo, vince chi ha il numero di PV più alto.

```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[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 (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).

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