Progetto Esame Programmazione I
a.a. 2018/2019

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

6 Dicembre 2018

Si realizzi un programma in linguaggio C che consiste dei seguenti tre file:

- `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 giocatori *Ninja* e *Ciccio* si sfidano a *Forsennait*, un gioco dove il territorio è pieno di insidie e si restringe dopo un certo numero di turni. Se sopravviveranno, si dovranno scontrare uno contro uno.

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)* crea mappa, *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 (stesso controllo su input per tutti gli altri menu stampati nel gioco). Nel caso la scelta sia 1, 2, o 3, chiamare la corrispondente funzione definita in *gamelib.c*, cioè *crea_mappa()*, *gioca()*, e *termina_gioco()*.

gamelib.h. Questo file deve contenere solamente le dichiarazioni delle funzioni definite in *gamelib.c* (come non *static*), e le definizioni dei tipi utilizzati nella libreria, cioè i tipi *struct Giocatore* e *struct Cella*, ed i tipi *enum Tipo_pericolo*, *enum Stato_giocatore* e *enum Tipo_oggetto*.

- *struct Giocatore* contiene *i)* un campo *enum Stato_giocatore stato*, *ii)* un campo che memorizza la posizione x sulla scacchiera, *iii)* un campo che memorizza la posizione y sulla scacchiera, e *iii)* un array zaino di lunghezza 4 che memorizza gli oggetti presi, *iv)* il numero di alieni uccisi.

- La *struct Cella* contiene i campi *i) enum Tipo_pericolo pericolo* e *ii) enum Tipo_oggetto oggetto*. Rappresentano il possibile pericolo e oggetto che si trova su una data cella.
- Il tipo *enum Tipo_pericolo* deve rappresentare i valori *nessun_pericolo, trappola, alieno*.
- Il tipo *enum Stato_giocatore* deve rappresentare i valori *vulnerabile, scudo_vita, solo_vita, solo_scudo*.
- Il tipo *enum Tipo_oggetto* deve rappresentare i valori *nessun_oggetto, medikit, pozione, materiale, colpi_lanciarazzi*.

gamelib.c. Questo file rappresenta il nucleo del progetto, dato che contiene l'implementazione delle funzioni che permettono di giocare. La prima struttura dati globale (e *static*) è rappresentata dalla scacchiera che rappresenta l' area di gioco: di essa non se ne conosce ancora la dimensione, quindi verrà rappresentata da un puntatore a *struct Cella *scacchiera= NULL*. La dimensione della scacchiera viene letta all'interno della funzione *crea_scacchiera()*, dove la scacchiera viene anche creata.

Esistono per lo meno altre due variabili globali *static struct Giocatore Ninja* e *static struct Giocatore Ciccio*, che rappresentano le statistiche dei due personaggi.

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

a) In questo punto, le tre funzioni che possono essere utilizzate dall'utente una volta chiamata la *crea_mappa()* nel menu principale.

   1) La funzione *crea_scacchiera()* viene richiamata da *main.c* e serve per creare la zona di gioco (scacchiera). Questa funzione richiede all'utente di inserire n, cioè la dimensione della scacchiera $n \times n$. Dopodiché la scacchiera deve essere allocata in memoria dinamica con *malloc(rows * cols * sizeof(struct Cella))*, ed assegnata al puntatore globale *scacchiera*. In seguito per accedere ad un elemento, si usa l'aritmetica dei puntatori con *scacchiera[i * cols + j]* (equivale a *scacchiera[i][j]*)[1]. Questa funzione richiede in input anche la probabilità (da 0 a 100) di ogni cella di contenere *nessun_pericolo*, di contenere una *trappola*, di contenere un *alieno* (per quanto riguarda i pericoli), e una probabilità per ogni oggetto possibile. Ricordarsi che una cella può contenere solo un pericolo e un oggetto (le probabilità si devono sommare a 100%: per esempio *nessun_pericolo* 60%, *trappola* 10%, *alieno* 30%). Infine, questa funzione scorre tutte le celle della scacchiera e, per ogni cella, utilizzando un numero random (vedi Figura 2) inizializza i campi della struttura.

      Infine, questa funzione sceglie casualmente su quale cella far iniziare ciascuno dei due giocatori (scrivendolo nella variabile *struct Giocatore* appropriata). I giocatori partono entrambi dallo stato *solo_vita*.
   2) La funzione *stampa_scacchiera()* permette di visualizzare il contenuto di tutte le celle (con tutti i loro campi), solo dopo aver creato la scacchiera.
   3) La funzione *termina_creazione()* permette di ritornare al menu iniziale (1-2-3 in *main.c*).

b) Una volta selezionata la funziona *gioca()* dal menu iniziale (possibile solo se è già stata creata una mappa), si comincia a giocare a turni: per esempio i turni dispari gioca *Ninja* e quelli pari *Ciccio* (oppure in maniera casuale far iniziale uno o l'altro). Ci sarà bisogno quindi anche di una variabile *unsigned int* che conti i turni passati. Di seguito, le mosse a disposizione in un turno di un giocatore; utilizzare una di queste mosse consuma il turno a disposizione del giocatore.

[1] `https://stackoverflow.com/questions/2128728/allocate-matrix-in-c`.

1) La funzione *muovi()* permette di spostarsi sulla scacchiera di una cella in una delle quattro direzioni. Chiedere al giocatore se si intende spostarsi a nord, est, sud, ovest. Gestire appropriatamente una mossa fuori dalla scacchiera (per esempio chiedendo di nuovo fino ad ottenere una mossa corretta). Quando un giocatore si sposta in una cella con una trappola, il giocatore automaticamente muore (e l'altro vince). Quando il giocatore si sposta in una cella con un alieno, può decidere se combattere oppure no: se non combatte, non può prendere l'oggetto eventualmente presente. Se combatte, uccide l'alieno automaticamente, ma ha il 35% di possibilità di perdere uno tra scudo (65%) o vita (35%). Se viene colpito allo scudo ed non lo ha, perde la vita (se in stato *solo_vita*, altrimenti vuol dire che è in stato *vulnerabile*, vedere dopo), se viene colpito alla vita e non ce l'ha (stato *solo_scudo*), viene eliminato lo scudo. Se viene colpito nello stato *vulnerabile*, il giocatore muore, e l'altro vince automaticamente. Il numero di uccisioni del giocatore viene incrementato di uno.
2) La funzione *usa_oggetto()* mostra al giocatore il contenuto dello zaino e chiede quale oggetto utilizzare (se ce ne sono). Gli effetti degli oggetti sono i seguenti: il *medikit* ristabilisce la vita (per esempio, da stato giocatore *vulnerabile* a *solo_vita*, oppure da *solo_scudo* a *scudo_vita*). La *pozione* ristabilisce lo scudo (alla stessa maniera). Gli altri due oggetti, *materiale* e *colpi_razzo* possono diventare utili nello scontro finale (vedere dopo). Lo zaino nella *struct Giocatore* può contenere massimo 3 elementi dello stesso tipo: in *zaino[medikit-1]* si trova il numero di *medikit* (da 0 a 3), ed in *zaino[colpi_lanciarazzi-1]* si trova il numero di *colpi_lanciarazzi* (da 0 a 3).

c) La funzione *termina_gioco()* permette di uscire dal gioco dopo aver mostrato le statistiche di vittorie tra i due giocatori.

Restringimento mappa gioco. Ogni t turni (decidere t in base ad n, cioè alla larghezza della scacchiera), la scacchiera si restringe di metà: invece che $n \times n$ diventa $(n/2) \times (n/2)$. Risolvere liberando (*free*) la precedente scacchiera e riallocarla (*malloc* o *calloc*) nuovamente della nuova dimensione, assegnando il nuovo puntatore sempre alla variabile *scacchiera*. Le posizioni dei due giocatori sulla nuova scacchiera sono scelte casualmente. La scacchiera viene nuovamente inizializzata con le stesse probabilità utilizzate nella *crea_scacchiera()* all'inizio del gioco.

Scontro finale. In qualunque istante i due giocatori si trovino nella stessa cella (accade per forza quando la scacchiera diventa 1×1), il gioco si interrompe e si passa alla fase di scontro finale. Ideare un algoritmo per risolvere la fase finale di uno contro uno, utilizzando sia le statistiche dei due personaggi (in *Struct Giocatore*), sia valori casuali (per esempio simulando il lancio di un dado). L'algoritmo deve prevedere comunque la creazione di una torre per giocatore, utilizzando una lista dinamica in cui si inserisce in coda un numero di piani pari al numero di *materiale* nello zaino (massimo 3 piani). Ogni *colpi_lanciarazzi* può togliere un piano (rimozione in coda). La costruzione di una torre più alta dell'altra avvantaggia il giocatore nell'algoritmo di vittoria. La *struct Piano* contiene due campi: un *unsignedshort int* che rappresenta il numero del piano (1,2,3) ed ovviamente un *struct Piano *prossimoPiano* che punta al piano successivo. Stampare il contenuto delle due torri subito dopo la creazione, e subito dopo aver sparato con il lanciarazzi. Un esempio in Figura. 1.

Il file *gamelib.h* deve essere incluso (*#include*) sia in *gamelib.c* che in *main.c*.

Ogni volta che un giocatore muore, l'altro automaticamente vince e la partita termina. Si ritorna al menu principale da dove è possibile creare una nuova scacchiera e tornare subito a giocare, oppure terminare il gioco stampando le statistiche di vittoria tra i due giocatori.

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

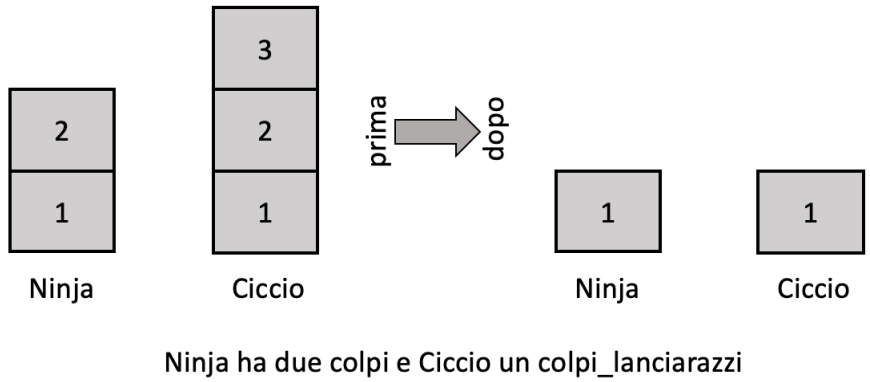

Figura 1: Esempio delle due torri dopo la costruzione e dopo l'uso di *colpi_lanciarazzi*.

`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 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 un punto aggiuntivo 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]. 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 il comportamento dell'algoritmo che risolve lo scontro finale.

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

```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 100 */
  printf("%d\n", rand() % 101); // Chiamare quando si ha bisogno di un numero random
}

```

Figura 2: Esempio di come funziona la generazione di un numero casuale.