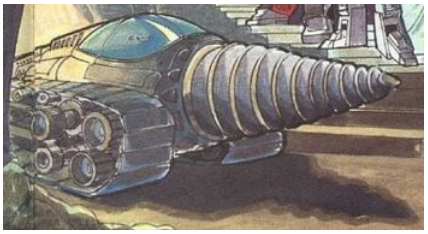

Progetto Esame Programmazione I
a.a. 2019/2020

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

10 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. La *Arvais* e la *Hartornen* sono due famiglie che si contendono da anni sul pianeta *Sabbie* l'estrazione della *Melassa*, una preziosissima sostanza fondamentale per la struttura della società galattica. L'estrazione è ostacolata da numerosi pericoli, come improvvise frane, scontri con la compagnia rivale, e i pericolosissimi *Bachi delle Sabbie*, che possono raggiungere lunghezze di centinaia di metri ed ingoiare gli escavatori in un col boccone.[1]

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. Il gioco viene giocato da due giocatori, che manovrano le scavatrici delle due famiglie. Le possibili scelte sono: *1)* crea cunicoli, *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_cunicoli()*, *gioca()*, e *termina_gioco()*. Una volta terminata la costruzione della mappa di gioco (cioè due cunicoli di caverne, uno per ciascun giocatore), si può costruirne un'altra (perdendo la prima: esiste una sola mappa giocabile), e non si può giocare senza prima aver costruito i due cunicoli. Il gioco è a turni, prima gioca un giocatore, poi l'altro.

[1] Si consiglia la lettura del romanzo *Dune* di Frank Herbert (`https://it.wikipedia.org/wiki/Dune_(romanzo)`) -E- poi la visione dell'omonimo film di David Lynch (`https://it.wikipedia.org/wiki/Dune_(film)`).

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 Scavatrice* è definita dalla sua posizione attuale all'interno delle caverna (un puntatore a *struct Caverna*), un serbatoio di energia da 0 a 4 unità di melassa (inizializzato a 4, quando a 0 la scavatrice non può più abbattere pareti, se scende sotto 0 viene distrutta ed il giocatore che la comanda perde il gioco), un serbatoio di raccolta da 0 a 10 unità di melassa.
- *struct Caverna* contiene tre puntatori a *struct Caverna*, i cui identificatori sono *avanti*, *sinistra*, *destra*. Se diversi da NULL, essi rappresentano le possibili direzioni in cui la scavatrice può continuare il proprio cammino senza distruggere una parte della caverna. Se uguali a NULL, la parete può essere solo abbattuta (vedi le funzioni successive). La caverna è definita anche da un campo *melassa* di tipo *enum Quantita_melassa* che descrive la quantità di melassa presente, e da un campo *imprevisto*. Infine contiene un campo *stato* di tipo *enum Tipo_caverna*.
- Il tipo *enum Tipo_imprevisto* deve rappresentare i valori *nessun_imprevisto, crollo, baco*.
- Il tipo *enum Quantita_melassa* deve rappresentare i valori *nessuna, poca, molta=3*.
- Il tipo *enum Tipo_caverna* deve rappresentare i valori *normale, speciale, accidentata, uscita*.

gamelib.c. Questo file rappresenta il nucleo del progetto: esso contiene quattro variabili globali, una *static struct Caverna* primo_cunicolo_arvais* (inizializzata a *NULL*) che punta alla prima zona della cunicolo della famiglia Arvais, e una *static struct Caverna* primo_cunicolo_hartornen* con le stesse caratteristiche, per la famiglia Hartornen. Il file contiene anche due variabili *static struct Scavatrice arvais* e *static struct Scavatrice hartornen*. Le variabili devono essere *static* perché modificabili solamente tramite le funzioni della libreria (solo all'interno di *gamelib.c*). Alla partenza, il serbatoio di energia delle due scavatrici è pieno, così come è vuoto quello di raccolta della melassa.

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

a) La funzione *crea_cunicoli()* 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_caverna()*, *canc_caverna()*, *stampa_cunicolo()*, e *chiudi_cunicoli()*. Si ricorda che devono essere creati, uno per volta, ciascuno dei due cunicoli per le due scavatrici. I due cunicoli NON si incrociano: le scavatrici giocano su percorsi separati. Tutte queste funzioni devono essere definite *static*, in quanto non devono essere chiamate da fuori *gamelib.c*. Le prime tre funzioni prendono come parametro un riferimento al giocatore per il quale stiamo creado la mappa di gioco.

   1) Inserimento di una caverna rispetto all'ultima caverna già creata (*ins_caverna()*). 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 avanti, a destra, a sinistra. Posso creare la prossima caverna soltanto lungo una delle tre direzioni. La prima inserzione, quando la lista è vuota, non distingue questa direzione (in pratica, esiste un solo puntatore ad inizio lista).

      Lo *stato* (di tipo *enum Tipo_caverna*) deve essere passato da tastiera durante la creazione. Nella fase di creazione, questo campo non può essere assegnato a *uscita*. L'uscita va trovata abbattendo le pareti delle caverne.

      La quantità di melassa rappresentata da *enum Quantita_melassa* è casuale con probabilità rispettivamente 50-30-20.

Il campo *imprevisto* di tipo *enum Tipo_imprevisto* è anch'esso inizializzato randomicamente, con probabilità rispettivamente 50-35-15.

2) Cancella l'ultima zona inserita nella mappa (*canc_caverna()*). Essa libera la memoria occupata dall'ultima caverna (*free()*).

3) Stampa i campi di tutte le caverne di uno dei cunicoli creato fino a quel momento (*stampa_cunicolo()*), compreso il valore del puntatore alla caverna successiva; ovviamente anche questa funzione prende come parametro un riferimento a uno dei due cunicoli.

4) Fine della creazione della mappa con *chiudi_cunicoli()*. 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 *crea_cunicoli()*. Non si può chiudere la mappa anche se ci sono meno di 10 caverne inserite in uno dei due cunicoli.

b) La funzione *gioca()* viene richiamata da *main.c* (punto 2). Le due scavatrici iniziano ciascuna all'inizio del cunicolo di pertinenza. Ogni turno è iniziato da uno dei due giocatori, poi seguito terminato dall'altro, in modo alternato. Ad ogni turno un giocatore sceglie dal menu della funzione *gioca()* UNA sola possibile mossa. Le possibili mosse sono rappresentate dalle funzioni *avanza()*, *abbatti()*, *aggira() esci()*. Queste funzioni sono tutte definite come *static* in *gamelib.c*. Ognuna delle seguenti funzioni "consumano" un turno di un giocatore.

1) con la funzione *avanza()* si avanza nell'unica direzione possibile (avanti/sinistra/destra), stampando anche tale direzione al giocatore che l'ha selezionata. Gli effetti di uno *stato* accidentato della nuova caverna sono quelli di consumare 1 unità di melassa dal serbatoio di energia. Uno stato speciale invece aggiunge 1 unità di melassa al serbatoio di energia (la caverna irradia melassa). L'effetto di un *imprevisto* crollo sono quelli di consumare 1 unità di melassa dal serbatoio di energia per potersi liberare dalle macerie. L'effetto di *imprevisto* baco porta allo svuotamento immediato di tutto il serbatoio di energia e serbatoio di raccolta di melassa (divorata dal baco). Se c'è melassa nella caverna, viene automaticamente raccolta, ma si deve chiedere al giocatore in quale serbatoio viene aggiunta (energia/raccolta). Tutte le volte che si avanza con la funzione *avanza()*, c'è una probabilità del 25% che non sia possibile a causa di un crollo improvviso; a questo punto deve essere invocata la funzione *aggira* (vedi dopo).

2) la funzione *abbatti()* può essere utilizzata per creare una nuova caverna in una delle direzioni non ancora aperte della caverna corrente. Costa 1 unità di energia. Con la funzione avanza si può quindi entrare nella nuova caverna abbattuta. La nuova caverna deve essere creata con *malloc()* in memoria dinamica ed ha le seguenti proprietà: la probabilità di *stato* di uscita aumenta (da 0) 5% per ogni turno passato (è del 50% al decimo turno). Accidentanta rimane sempre al 20%, speciale/normale si dividono la restante parte a metà (per esempio, al primo turno 100-5-20% = 75% diviso a metà tra speciale e normale). La quantità di melassa della nuova caverna è rispettivamente 40-40-20 (nessuna/poca/molta). Le probabilità del campo *imprevisto* della nuova caverna sono 40-40-20 (nessun_imprevisto, crollo, baco).

3) La funzione *aggira()* inserisce una nuova caverna (funzione *malloc*) tra la corrente e quella successiva per la quale si è verificato un crollo imprevisto a seguito dell'invocazione della funzione *avanza()* (vedi precedente funzione). Può essere chiamata -solo- quando accade questo evento. Le probabilità per tutti i campi sono le stesse di quando viene creata una caverna durante la fase di creazione dei cunicoli (vedi funzione *ins_caverna()*). La logica di questa funzione corrisponde ad una inserzione in mezzo alla lista. La scavatrice si muove anche immediatamente dentro la nuova caverna creata.

4) La funzione *esci()* serve per uscire dal cunicolo, e può essere chiamata solamente in una caverna con *stato* uguale a uscita.

Se una scavatrice scende ad un livello di energia nel suo serbatoio minore di 0, il giocatore corrispondente perde e l'altro vince. Se tutte e due le scavatrici riescono ad uscire dal gioco, vince il giocatore che ha raccolto più melassa (nel serbatoio di raccolta).

Per ogni scavatrice, esiste una probabilità che aumenta del 3% ogni turno (partendo da 3% al turno 1) di incrociare il percorso dell'altra scavatrice. In questo caso definire il codice di scontro diretto tra le due a piacere, descrivendolo anche nel file README del progetto Github (vedere anche le note finali). Il tipo *struct Scavatrice* (e altro) pùo essere esteso in modo da gestire lo scontro come desiderato.

Alla fine dello scontro si può invocare la funzione *termina_gioco()*, che dealloca tutto ciò che è stato memorizzato nella memoria dinamica, scorrendo entrambi i cunicoli ed utilizzando la funzione *free()* su ogni puntatore di ogni caverna. Infine saluta il giocatore stampando un messaggio di "Game Over". Altrimenti si può ricreare una nuova rete di cunicoli (*crea_cunicoli()*) e continuare poi a giocare con *gioca()*.

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 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 99 */
  printf("%d\n", rand() % 100); // Chiamare quando si ha bisogno di un numero random
}

```

Figura 1: Esempio di come funziona la generazione di un numero casuale.