APPELLO SCRITTO DI PROGRAMMAZIONE 1
CORSO DI LAUREA IN MATEMATICA
UNIVERSITÀ DEGLI STUDI DI MILANO

2020–2021

18.06.2021

Indice

Avvertenza. Il tema d'esame richiede lo sviluppo di un singolo programma che implementi una versione semplificata del gioco Forza 4. Le definizioni precise e i dettagli necessari sono dati nel testo. Svilupperete delle funzioni ausiliarie nel risolvere gli esercizi. Integrerete poi tutte le funzioni ausiliarie in un unico programma finale, che sarà costituito da un solo file sorgente. (Il tema d'esame richiede esplicitamente l'implementazione di alcune funzioni ausiliarie. È comunque sempre ammissibile aggiungere altre funzioni ausiliarie qualora lo riteniate opportuno per una migliore implementazione della vostra soluzione.) Si raccomanda di leggere interamente il tema d'esame con attenzione prima di cominciare a scrivere le soluzioni dei singoli esercizi, in modo da avere chiaro l'obiettivo finale.

Consegnerete solo codice che compili senza errori tramite il sito: `https://upload.di.unimi.it/`, dopo aver fatto login usando le credenziali universitarie.

Premessa: descrizione del gioco.

Forza 4 si gioca su una matrice di 6 righe per 7 colonne. I due giocatori muovono a turno. Il giocatore $i = 1, 2$ sceglie una colonna della matrice e inserisce una pedina di colore i in quella colonna. Per la forza di gravità, la pedina discende fino alla prima riga non ancora occupata (da qui la prima parte del nome del gioco). Se la colonna è già pienamente occupata il giocatore i non potrà fare la sua mossa su questa colonna, e dovrà sceglierne un'altra. Si veda la Fig. 1.

Ultima revisione: 18 giugno 2021.

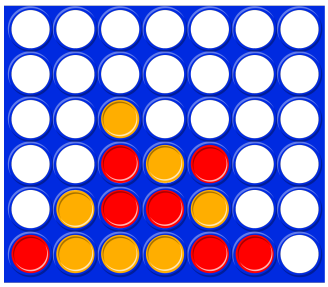

FIGURA 1. Il gioco Forza 4.

Scopo del gioco è allineare $n = 4$ pedine dello stesso colore consecutivamente in verticale, orizzontale oppure in diagonale (da qui la seconda parte del nome del gioco). Vince il giocatore che per primo riesce ad allineare le proprie pedine in questo modo. Se il campo di gioco è pieno prima che uno dei due giocatori abbia vinto, la partita termina in parità.

Nota 1. Per semplificare l'implementazione, assumeremo che le uniche configurazioni vincenti siano quelle in orizzontale e in verticale.

Nota 2. Il programma implementerà una versione generalizzata di Forza 4, chiamata Forza n, in cui sia la dimensione $r \times c$ e il numero di pedine da allineare n sono passati come argomenti del main tramite linea di comando.

Dovrete quindi scrivere un programma in C che permetta all'utente di giocare a Forza n contro il computer. Assumete per semplicità che sia sempre l'utente a fare la prima mossa di una nuova partita. Ancora per semplificare, le mosse del computer saranno casuali. Per generare numeri pseudo-casuali, per esempio un numero intero `int x` in $\{0, 1, \ldots, k-1\}$, usate le funzioni `int rand()`, `void srand(int)` e l'operatore `%` di modulo in questo modo:

```
srand(time(NULL)); //inizializzo un seed
x=rand()%k; // genero il numero casuale x
```

Per usare queste funzioni dovrete includere `stdlib.h` e `time.h`.

Nota Bene. Passare parametri come input della funzione `main` da linea di comando richiede di sfruttare le variabili che sono parametri formali della funzione `main`:

```
main(int argc, char* argv[])
```

ESERCIZIO 1

Creazione campo di gioco, visualizzazione e controlli.

Punti: 12.

Utilizzando i tre parametri n, r, e c passati tramite linea di comando come input del metodo main, dichiarate una matrice globale `G` di interi di dimensione $r \times c$ che memorizzerà il campo di gioco. Conveniamo che il giocatore 1 sia l'utente e che il giocatore 2 sia il computer. Conveniamo che $n > 0$ è il numero di pedine che vanno allineate per arrivare alla vittoria, $r \times c$ è la dimensione del campo, `G[i][j]` varrà 0, 1 o 2 a seconda che la casella (i, j) sia vuota (0), contenga una pedina dell'utente

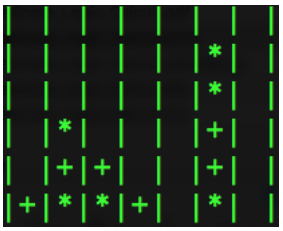

FIGURA 2. Visualizzazione sul terminale del campo di gioco.

(1), o contenga una pedina del computer (2). Conveniamo che le pedine dell'utente siano rappresentate a video da `+`, e quelle del computer da `*`. Il campo di gioco sarà quindi visualizzato sul terminale in modo simile a quanto mostrato in Fig. 2.

Scrivete le seguenti funzioni.

- `void show()` — visualizza il campo di gioco.
- `int game_over()` — restituisce `-1` se il campo di gioco corrisponde a una partita ancora in corso, `0` se corrisponde a una partita terminata e patta, `1` se corrisponde a una partita vinta dall'utente, e `2` se corrisponde a una partita vinta dal computer. Assumete che i valori contenuti nella matrice `G` siano tali che esattamente una di queste condizioni si verifichi.

Per implementare la funzione `int game_over()` dovrete scandire la matrice `G` nei modi appropriati. Ricordate che non è necessario controllare se vi siano configurazioni vincenti in diagonale: cfr. la Premessa. *Suggerimento.* Potete implementare questa procedura così: prima eseguite i controlli necessari per verificare che uno fra l'utente o il computer abbiano realizzato una configurazione vincente; se questo è il caso, restituite il controllo alla procedura chiamante con valore appropriato; altrimenti, eseguite i controlli necessari per verificare che la partita sia in corso, oppure sia terminata con un pareggio, e restituire il controllo alla procedura chiamante con valore appropriato.

ESERCIZIO 2

Esecuzione del gioco: prima versione della procedura `main`.

Punti: 18.

Scrivete un programma che, all'avvio, permetta all'utente di giocare una partita di Forza n contro il computer. L'utente ha sempre la prima mossa. Quando è il suo turno, l'utente specifica la colonna $j = 1, \ldots, c$, nella quale vuole inserire la pedina. Se la colonna è interamente occupata, il programma forza il reinserimento della scelta finché essa non risulti ammissibile. Quando è il turno del computer, il programma invoca una procedura ausiliaria di nome `mossa` e prototipo appropriato che restituisce un numero intero pseudo-causale compreso fra 1 e c. Se la colonna corrispondente è interamente occupata, il programma chiama nuovamente la procedura `int mossa()` fino a quando il valore restituito non corrisponda ad una colonna ammissibile. Dopo ciascuna mossa il programma visualizza il campo di gioco aggiornato e controlla lo stato della partita con la procedura `game_over` dell'Esercizio 1. Il programma prosegue l'esecuzione del gioco oppure interrompe la partita, annunciando l'esito in modo appropriato, a seconda del valore restituito da

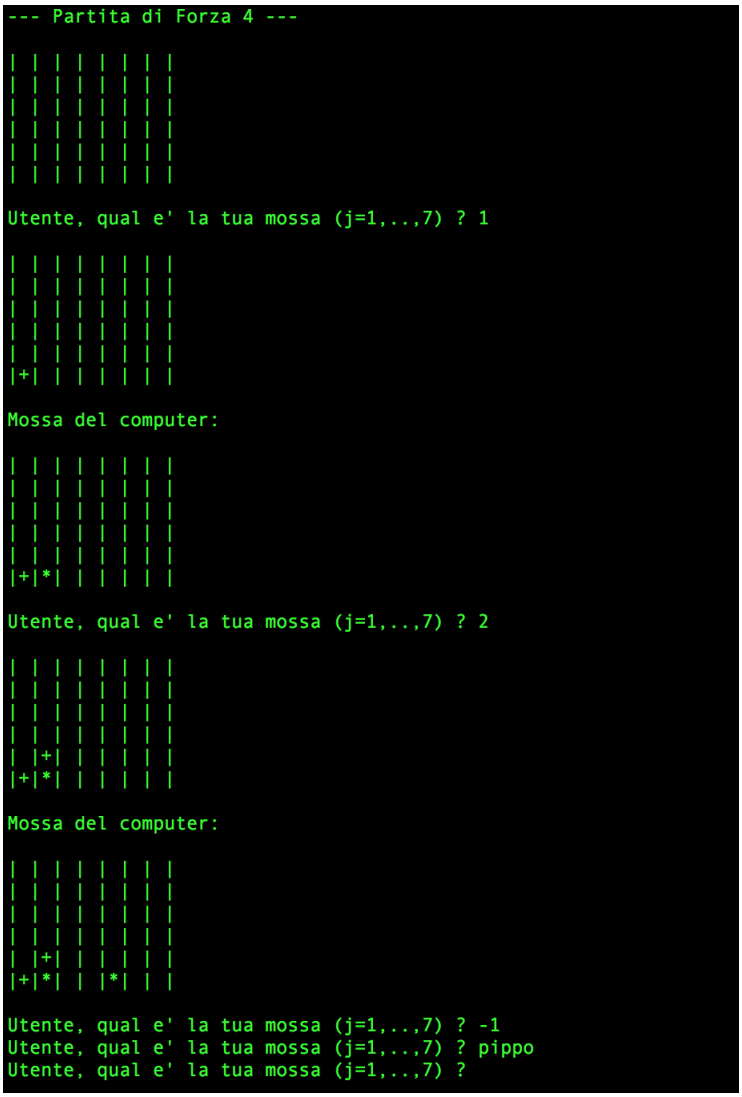

FIGURA 3. Esempio di esecuzione del programma dell'Esercizio 2.

`game_over`. Nell'implementare l'esecuzione del gioco, usate due funzioni ausiliarie seguenti.

- `int leggi_mossa()` — legge da standard input un numero inserito dall'utente che rappresenta la colonna in cui l'utente vuole inserire la pedina. La funzione restituisce la mossa dell'utente (cioè, l'intero $j = 1, \ldots, c$).
- `int scrivi(int gioc, int j)` — tenta di inserire la pedina nella colonna j da parte del giocatore `gioc` (cioè, di scrivere il valore appropriato nella matrice `G`). Se l'esecuzione della mossa non riesce (cioè, se la j-esima colonna del campo di gioco è interamente occupata) restituisce 0, altrimenti 1.

Per un esempio di esecuzione del programma si veda la Fig. 3.

Istruzioni per la consegna

- Funzioni da consegnare (in un unico file sorgente).

  `main`

  `show`

  `game_over`

  `mossa`

  `leggi_mossa`

  Tutte le eventuali altre funzioni ausiliarie che abbiate scritto.

- Sito per la consegna:

  `https://upload.di.unimi.it`

- Autenticarsi con le proprie credenziali di posta elettronica d'ateneo.
- Consegnare solo il codice sorgente (file `*.c`), non l'eseguibile.

(V. Marra) Dipartimento di Matematica *Federigo Enriques*, Università degli Studi di Milano, via Cesare Saldini, 50, I-20133 Milano
Email address: vincenzo.marra@unimi.it