



Dipartimento di Informatica "Giovanni Degli Antoni"

Corso di Laurea Triennale in Informatica

Marco Tarini - Università degli Studi di Milano

Matteo Luperto- Università degli Studi di Milano

Architettura degli Elaboratori II Laboratorio

Progetto di gruppo: «Forza 4»

Progetto di gruppo: forza 4

- Un piccolo progetto in MIPS che consente a 2 giocatori di fare una partita al gioco «forza 4»
 - Multiplayer «hot-seat»
- Concordiamo le strutture dati utilizzate, e una suddivisione in funzioni del problema
- Ogni funzione verrà implementata da studenti in modo indipendente
 - E' importante seguire tutte le convenzioni del MIPS, affinché le funzioni possano interagire correttamente
 - Ogni file deve esporre (attraverso `.globl`) le funzioni concordate

plancia

(etichetta globale)
array **9** x **7** di BYTE

Semantica:

0: vuoto .
1: giocatore **X**
2: giocatore **O**
3: pieno #
(ai bordi sx, dx,
e basso)

```
# . . . . . #
# . . . . . #
# . . . . . #
# . . . . . #
# . . . . . #
# . . . . . #
# # # # # # # #
```

initPlancia

input: nessuno
output: nessuno

(Re-)setta la plancia
con i valori di inizio
partita (bordi compresi!)

main

Questo file alloca la plancia nei dati
statici e la espone agli altri file!

Sequenza (infinita) di partite:

In ogni partita:
si alternano le mosse dei giocatori,
si mostra la plancia dopo ogni mossa,
fino a fine partita (vittoria o
terminazione).

chiediMossa

input: giocatore (1 o 2)
output: la mossa giocata,
o -1 per fine partita

Per immettere la mossa, si preme un tasto
da 1 a 7, oppure «f» per fine partita.
Fa ripetere le mosse invalide.
La mossa è invalida se la colonna
giocata è piena, o indice colonna invalido.

giocaMossa

input: colonna (numero da 1 a 7),
giocatore (1 o 2)
output: colonna piena o no (un bool)

Aggiorna la plancia (a meno che la colonna
sia piena).



disegnaPlancia

input: nessuno
output: nessuno

Disegna la plancia (per es, in
Ascii Art).

checkVittoria

input: nessuno
output: 1 o 2 se quel giocatore ha
vinto, 0 altrimenti.

plancia

(etichetta globale)
array **9** x **7** di BYTE

Semantica:

0: vuoto .
1: giocatore **X**
2: giocatore **O**
3: pieno #
(ai bordi sx, dx,
e basso)

```
# . . . . . #
# . . . . . #
# . . . . . #
# . . . . . #
# . . . . . #
# . . . . . #
# # # # # # # #
```

initPlancia

input: nessuno
output: nessuno

(Re-)setta la plancia
con i valori di inizio
partita (bordi compresi!)

main

Questo file alloca la plancia nei dati
statici e la espone agli altri file!

Sequenza (infinita) di partite:

In ogni partita:

*si alternano le mosse dei giocatori,
si mostra la plancia dopo ogni mossa,
fino a fine partita (vittoria o
terminazione).*

chiediMossa

input: giocatore (1 o 2)
output: la mossa giocata,
o -1 per fine partita

*Per immettere la mossa, si preme un tasto
da 1 a 7, oppure «f» per fine partita.
Fa ripetere le mosse invalide.
La mossa è invalida se la colonna
giocata è piena, o indice colonna invalido.*

giocaMossa

input: colonna (numero da 1 a 7),
giocatore (1 o 2)
output: colonna piena o no (un bool)

*Aggiorna la plancia (a meno che la colonna
sia piena).*

0	1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16	17
18	19	20	21	22	23	24	25	26
27	28	29	30	31	32	33	34	35
36	37	38	39	40	41	42	43	44
45	46	47	48	49	50	51	52	53
54	55	56	57	58	59	60	61	62

disegnaPlancia

input: nessuno
output: nessuno

*Disegna la plancia (per es, in
Ascii Art).*

checkVittoria

input: nessuno
output: 1 o 2 se quel giocatore ha
vinto, 0 altrimenti.

sgiocaMossa

input: colonna (numero da 1 a 7),

output: nessuno

*Toglie la pedina più in alto dalla colonna
specificata (si suppone che ci sia: nessun check è necessario)*

valutaScacchiera

input: nessuno

output: «valore» della scacchiera

*Il valore tanto più **negativo** quanto più il giocatore **1** pare in vantaggio,
e tanto più **positivo** quanto più il giocatore **2** pare in vantaggio.*

Se il giocatore 1 ha vinto, il valore è -10^9

Se il giocatore 2 ha vinto, il valore è $+10^9$

Altrimenti, qualche valore intermedio.

*Come: testare tutte le possibili file di 4 caselle (oriz, vert, diago 1, diago 2).
Per ogni fila: se è tutta vuota, non contribuisce al «valore»;
se contiene N pedine di uno dei due giocatori e il resto è vuoto,
contribuisce $N*N$ punti al «valore», in positivo o in negativo.*

trovaMossaMiglioreAI

input: per quale dei due giocatori (1 o 2)

output: (1) la colonna sulla quale è meglio giocare (zero se nessuna mossa è possibile);

(2) il valore della scacchiera ottenuta con quella giocata

Algoritmo (semplice):

per ogni possibile mossa...

*(a) Tentare di giocarla (con **giocaMossa**)*

*(b) Se la mossa è possibile, allora valutare la scacchiera risultante (con **valutaScacchiera**).*

*(c) Sgiocare la mossa appena effettuata (con **sgiocaMossa**).*

In questo ciclo, tenere taccia della mossa migliore e del suo valore.

*Alla fine, riportare la mossa migliore (quella con valore massimo o minimo, a seconda del
giocatore e il valore corrispondente).*

Trucchetti!

*Se **P** è il valore di un player (1 o 2)*

allora:

***3-P** vale... l'altro giocatore*

P*2-3** vale... **-1** , quando **P = 1

+1** , quando **P = 2

giocaMossa

input: colonna (numero da 1 a 7),
giocatore (1 o 2)

output: mossa valida o no (bool)

checkVittoria

input: nessuno

output: 1 o 2 se quel giocatore ha vinto,
0 altrimenti.

sgiocaMossa

input: colonna (numero da 1 a 7),

output: nessuno

valutaScacchiera

input: nessuno

output: «valore» della scacchiera

chiediMossa

input: giocatore (1 o 2)

output: la mossa giocata, o -1 per fine

disegnaPlancia

input: nessuno

output: nessuno

initPlancia

input: nessuno

output: nessuno

trovaMossaMiglioreRicorsivo

input: (1) per quale dei due giocatori (1 o 2)
(2) profondità! Valore da 0 (scemo ma veloce) a ... tanto (furbo ma lento)
output: (1) la colonna sulla quale è meglio giocare
(zero se nessuna mossa è possibile);
(2) il valore della scacchiera ottenuta con quella giocata

Algoritmo (semplice):
per ogni possibile mossa...
(a) Tentare di giocarla (con **giocaMossa**)
(b) Se la mossa è possibile, allora valutare la scacchiera risultante ...
(con **valutaScacchiera**).

SE HAI VINTO: restituire la mossa (e il valore $\pm 10^5$)
funzione **checkVittoria**). [CASO BASE 1]
ALTRIMENTI: se profondità è 0, con **valutaScacchiera** [CASO BASE 2]
altrimenti, con questa stessa funzione [CASO RICORSIVO]
a profondità attuale - 1
(pensare agli altri parametri)

(c) Sgiocare la mossa appena effettuata (con **sgiocaMossa**).

In questo ciclo, tenere taccia della mossa migliore e del suo valore.
Alla fine, riportare la mossa migliore (quella con valore massimo o minimo, a seconda
del giocatore).

main

Sequenza (infinita) di partite:
In ogni partita: chiedere quanti giocatori (0, 1, o 2) ...

Trucchetti!
Se **P** è il valore di un player (1 o 2)
allora:
 3-P vale... l'altro giocatore
 P*2-3 vale... **-1** , quando **P = 1**
 +1 , quando **P = 2**

giocaMossa

input: colonna (numero da 1 a 7),
giocatore (1 o 2)
output: mossa valida o no (bool)

checkVittoria

input: nessuno
output: 1 o 2 se quel giocatore ha vinto,
0 altrimenti.

sgiocaMossa

input: colonna (numero da 1 a 7),
output: nessuno

valutaScacchiera

input: nessuno
output: «valore» della scacchiera

chiediMossa

input: giocatore (1 o 2)
output: la mossa giocata, o -1 per fine

disegnaPlancia

input: nessuno
output: nessuno

initPlancia

input: nessuno
output: nessuno

Perché funziona

- È un algoritmo «min-max»
- Simula in profondità N mosse da parte dei due giocatori
- Lo scopo è *minimizzare* la *massima* perdita possibile (se lascio al mio avversario una mossa buona da fare, la farà; voglio dargli meno opzioni buone possibili per il prossimo turno; gioco in maniera cauta/conservativa).

