Campo Minato

di Zanatta Giacomo – 859156

1. COMPILAZIONE

I sorgenti si trovano nella cartella *campominato.* Per compilare, eseguire da terminale:

```
1. gcc -g -std=gnu89 -Wall -pedantic -c campo.c -o campo.o
2. gcc -g -std=gnu89 -Wall -pedantic -c gioco.c -o gioco.o
3. gcc -g -std=gnu89 -Wall -pedantic -c listcoord.c -o listcoord.o
4. gcc -g -std=gnu89 -Wall -pedantic -c mosse.c -o mosse.o
5. gcc -g -std=gnu89 -Wall -pedantic -c main.c -o main.o
6. gcc -g -std=gnu89 -Wall -pedantic -c inout.c -o inout.o
7. gcc  -std=gnu89 -Wall -pedantic campo.o  gioco.o  listcoord.o  mosse.o  main.o  inout.o -o
campominato
```

Oppure utilizzare il makefile con il comando make.
Per giocare eseguire il file *campominato*

2. CAMPO DI GIOCO

Una cella del campo di gioco è una struttura contente tre valori interi: *value*, che contiene il valore di una cella, *marcata*, che è pari a 1 se la cella è stata marcata e 0 se non è marcata) e *scoperta*(0 se la cella non è stata scoperta, >0 se la cella è stata scoperta).
Value è codificato nel seguente modo: -1 se la cella contiene una bomba, N (0<=N<=8) se la cella è circondata da N bombe.
Il campo di gioco è una matrice di celle NxM, con N>1 e M>1 definiti dall'utente.

3. FUNZIONALITÀ

Il programma permette di:

- **generare uno schema**
- **leggere uno schema da file**
- **scrivere uno schema su file**
- **giocare in modo interattivo**

4. GENERAZIONE DI UNO SCHEMA

Quando si andrà a generare uno schema, verrà chiesto all'utente di inserire le dimensioni del campo, il numero di bombe (intero compreso tra 1 e (lunghezza*larghezza)-1)
Verrà inoltre richiesto di inserire il numero massimo di annullamenti.
Gli annullamenti vengono gestiti mediante un vettore circolare di coordinate (int x, int y) di dimensione (max_mosse*(max_mosse-1)/2)+1. Quando si effettuerà una mossa, le coordinate della mossa verranno inserite in coda a questo vettore, e se si dovranno annullare n mosse verrano annullate le ultime n mosse del vettore (Politica LIFO). Ho preferito usare un vettore circolare in quanto posso gestire con comodità il numero di massimo di mosse annullabili (pari alla dimensione del vettore), e se il vettore è pieno posso tranquillamente andare a sostituire il primo elemento in ordine di inserimento, in quanto sono certo che quella mossa non verrà mai annullata.
Le bombe vengono inserite in modo casuale nel campo, e all'inserimento di una bomba verrà incrementato di 1 il valore delle celle limitrofe (se queste celle non sono anch'esse una bomba).

5. LETTURA DI UNO SCHEMA

Quando si vuole leggere uno schema da un file, all'utente verrà chiesto di inserire un nome di un file, e se il file esiste verrà letto. I primi due valori interi incontrati saranno le dimensioni del campo di gioco, mentre i successivi valori saranno le coordinate dove si andranno ad inserire le bombe.
Viene usata una lista d'appoggio, dove verranno memorizzate le coordinate delle bombe.
Ho preferito usare una lista di coordinato invece che scrivere direttamente nella matrice per evitare di

scrivere una funzione che gestisce troppe cose: la funzione che legge uno schema si occuperà solo di leggere uno schema, sarà compito di un'altra funzione gestire l'alloccamento del campo e l'inserimento delle bombe.

6. SCRITTURA DI UNO SCHEMA

Dopo aver creato un campo, verrà chiesto all'utente se desidera salvare il campo su un file.
Se la risposta è affermativa, verranno memorizzate su un file le dimensioni del campo e le coordinate delle bombe. Il nome del file è definito dall'utente, e se il file esiste già verrà chiesto all'utente se desidera sostituire il file.

7. GIOCARE A CAMPO MINATO

Dopo aver generato un campo, o dopo aver letto un campo da un file, sarà possibile iniziare una partita.
Il campo viene stampato su stdout, e le celle verranno visualizzate nel seguente modo:

- **$ → cella marcata**
- # → **cella coperta**
- **0,1,2,3,4,5,6,7,8 → cella scoperta,** il numero stampato sarà il suo valore (ossia il numero di bombe limitrofe)
- **B** → cella scoperta, **bomba**.

L'utente può: effettuare una mossa, marcare/smarcare una cella, uscire dal gioco.

1. EFFETTUARE UNA MOSSA: verrà chiesto all'utente di inserire le coordinate della cella che vuole scoprire. Se la cella è marcata o è stata già scoperta verrà stampato un errore. Se la cella è una bomba, verrà chiesto all'utente di poter usare un annullamento. La partita finisce quando il giocatore ha beccato una bomba e non ha più annullamenti o quando il giocatore vince, ossia ha scoperto DIM-BOMBE celle.
2. MARCARE/SMARCARE UNA CELLA: l'utente può smarcare o marcare una cella, inserendo le coordinate. Se la cella non è stata già scoperta ed è marcata, allora verrà smarcata. Se invece la cella è smarcata (e non è stata scoperta) verrà marcata.
3. USCIRE DAL GIOCO: all'uscita del gioco verranno liberate le porzioni di memoria dinamica utilizzate dal programma, e l'utente verrà portato al menù principale.

8. FUNZIONALITÀ EXTRA (INTELLIGENZA ARTIFICIALE)

Ho implementato una semplice intelligenza artificiale che risolve uno schema. I sorgenti si trovano nella cartella *campominato-extra*, e si compilano usando il makefile oppure nel seguente modo:

```
1. gcc -g -std=gnu89 -Wall -pedantic -c campo.c -o campo.o
2. gcc -g -std=gnu89 -Wall -pedantic -c gioco.c -o gioco.o
3. gcc -g -std=gnu89 -Wall -pedantic -c listcoord.c -o listcoord.o
4. gcc -g -std=gnu89 -Wall -pedantic -c mosse.c -o mosse.o
5. gcc -g -std=gnu89 -Wall -pedantic -c main.c -o main.o
6. gcc -g -std=gnu89 -Wall -pedantic -c inout.c -o inout.o
7. gcc -g -std=gnu89 -Wall -pedantic -c ia.c -o ia.o
7. gcc  -std=gnu89 -Wall -pedantic campo.o  gioco.o  listcoord.o  mosse.o  main.o  inout.o ia.o
-o campominato
```

8.1 FUNZIONAMENTO DELL'IA

L'IA segue i seguenti passi:
1. inizializza una struttura, composta da: bombe rimanenti nel campo, celle scoperte, matrice RIGHExCOLONNE della dimensione del campo di gioco, probabilità (che contiene la probabilità casuale di beccare una bomba).
Ogni cella della matrice dell'IA è composta da: prob di beccare una bomba, probabilità casuale, prima_volta (identifica la prima volta che la cella è stata toccata), scoperta (identifica se la cella è già stata scoperta o meno), celle vicine coperte (identifica il numero di celle limitrofe alla cella che non sono ancora state scoperte), e bombe limitrofe non scoperte (identifica il numero di bombe limitrofe alla cella che devono ancora essere state scoperte.
2. cerca la cella che ha Probabilità minore nel campo e che non è ancora stata scoperta
3. esegue la mossa: scopre la cella nel campo. Se è una bomba, allora viene incrementato celle scoperte, e viene segnalato alle celle limitrofe che è stata scoperta una cella e che quella cella è una bomba. Se ci sono annullamenti, allora la mossa viene annullata e quella cella viene marcata. Se non ci sono, ha perso.

Se non è una bomba, allora viene segnalato alle celle limitrofe che si è scoperta una cella, e se il suo valore è maggiore di 0 allora viene modificata la Probabilità che quella cella sia una bomba.
4. si continua ad eseguire mosse finché il campo è stato scoperto (VITTORIA) o quando non si hanno più annullamenti (SCONFITTA).

Dato che campo minato è un gioco aleatorio (possono quindi verificarsi dei casi in cui bisogna andare per tentativi, ad esempio il primo), non è detto che il computer riesca sempre a risolvere uno schema.

8.2 PROBABILITÀ DI UNA CELLA

Se la probabilità (X) di una cella è 1, allora quella cella conterrà una bomba (viene marcata).
Se la probabilità di una cella è 0, allora quella cella sicuramente non conterrà una bomba.
Se, invece, la probabilità e compresa tra 0 e 1, quella cella può contenere una bomba.
La cella da scoprire viene scelta tra le celle che hanno probabilità minore.