Minesweeper

Il campo di gioco consiste in un rettangolo (o quadrato) di *m* * *n* celle.
Nel classico gioco del campo minato se una cella contiene una mina, quando sarà cliccata la mina esploderà e farà terminare il gioco. Se il quadrato non contiene una mina, possono accadere due eventi:

- se appare un numero esso indica la quantità di quadrati adiacenti (inclusi quelli in diagonale) che contengono mine;
- se non appare nessun numero il gioco ripulisce automaticamente i quadrati adiacenti a quello vuoto (fino a quando non conterranno un numero).

Si vince la partita quando tutti i quadrati che non contengono mine saranno scoperti.
In questa versione per MiniZinc viene inserito come input la griglia di partenza arricchita con i numeri (hint) che indicano la quantità di quadrati adiacenti che contengono mine. Le celle restanti (celle sconosciute) possono essere vuote o mine.
Il problema del Minesweeper è un CSP (Constraint Satisfaction Problem) e l'obiettivo è di individuare e stampare la griglia delle mine trovate, come un matrice di 1 (cella con mina) e 0 (cella senza mina).
Il set-up del programma prevede in input il numero di righe (*r*), di colonne (*c*) e di mine (*nmines*) da cercare e la griglia di partenza (*game*). La griglia è data da un array a due dimensioni di *r* righe e *c* colonne, ogni cella è composta da un intero o da *X* che è una costante per il numero -1 e indica le celle sconosciute (si è evitato di mettere -1 direttamente, o un altro numero, per non fare confusione visualizzando la griglia).
I vincoli principali sono:

- il numero di mine trovate deve essere uguale a quello inserito come input
- data una cella con un numero maggiore di X (-1), il numero indica le probabili mine nell'intorno della cella
- se nella cella c'è un numero maggiore di X (-1) non può esserci una mina

Il primo vincolo, operativamente, è stato costruito come una somma sull'intera matrice di mine trovate: se la somma è uguale al *nmines* allora è soddisfatto.
Il secondo è il terzo vincolo sono all'interno di un ciclo *for* che scorre la matrice di partenza (*game*): se il contenuto della cella è un numero maggiore di *X* allora non c'è una mina e va controllato l'intorno della cella alla ricerca di mine.
La propagazione di questi tre vincoli risolve il gioco.
Il secondo vincolo è il cuore del programma ed è sufficiente a risolvere il problema, infatti, eliminando il terzo, che non porta a nessuna nuova propagazione, si ottengono miglioramenti di tempo nell'esecuzione del programma (attorno al 50%) mentre il primo non porta a nessun significativo cambiamento nelle prestazioni ma è importante per la correttezza logica del modello.
Invertendo anche gli ultimi due vincoli si hanno cali di prestazioni notevoli perché il secondo vincolo è molto più restrittivo ed esegue un filtering maggiore sulla matrice.
Al fine di ottimizzare la soluzione si sono utilizzate delle annotazioni a *solve* applicate alla matrice delle soluzioni da cercare. Le annotazioni impostano delle strategie che devono essere applicate nel trovare la soluzione: *first_fail* sceglie il dominio più ristretto della variabile, *indomain_min* sceglie il valore più basso e *complete* imposta la ricerca sull'intero albero.