Introducción

El ejercicio propuesto a resolver es el siguiente puzzle (puesto que desconozco el nombre)

Tenemos un tablero cuadrado (no es necesario, la extensión que se ha hecho soporta matrices no cuadradas) con la siguiente configuración inicial:

1	0	0
1	0	0
1	0	0

(No necesariamente el tablero tiene que ser de 3x3. La configuración inicial puede ser cualquier combinación de 0s y 1s)

Y el objetivo es, operando sobre filas, columnas y diagonal cuya aplicación sobre el tablero invierte los valores abarcados, llegar al estado final:

1	1	1
1	1	1
1	1	1

Donde todos los elementos son 1s

Para un mejor entendimiento expondremos una pequeña traza de ejemplo:

1. Estado inicial

1	1	0
0	0	1
0	0	1

2. Aplica operador fila sobre fila 1

0	0	1
0	0	1
0	0	1

3. Aplica operador columna sobre columna 2

0	1	1
0	1	1
0	1	1

4. Aplica operador columna sobre columna 2

1	1	1
1	1	1
1	1	1

Búsqueda en espacio de estados

Vamos a modelizar el problema mediante una búsqueda en espacio de estados y usando el módulo *busqueda.pl* vamos a diseñar un programa Prolog que resuelva el dicho problema.

Extenderemos las diversas búsquedas para construir el estado inicial y final a partir de una matriz de entrada.

```
:- use_module(busqueda).
:- dynamic estado_inicial/1.
:- dynamic estado_final/1.

profundidad_con_ciclos_2([H|T],S) :-
                  retractall(estado_inicial(_)),
                 retractall(estado_final(_)), assert( estado_inicial([H|T]) ),
                 length([H|T],F), length(H,C), matriz1(F,C,EF),
                 assert(estado_final(EF)),
                 profundidad_con_ciclos(S).

anchura_2([H|T],S) :- retractall(estado_inicial(_)),
        retractall(estado_final(_)), assert( estado_inicial([H|T]) ),
        length([H|T],F), length(H,C), matriz1(F,C,EF),
        assert(estado_final(EF)),
        anchura(S).
```

Los estados los representaremos mediante una lista de listas (matriz) cuyo tamaño dependerá del problema (estado inicial)

```
%% Estado inicial

%% [[1,0,0]
%%  [1,0,0]
%%  [1,0,0]]

%% Estado final (para 3x3)
%% [[1,1,1]
%%  [1,1,1]
%%  [1,1,1]]
```

Operadores

Para el cálculo de sucesores usaremos 3 operadores:

- Operador fila (para cada una de las filas). Invierte los valores de la fila seleccionada

1	0	0
1	0	0
1	0	0

Aplica operador fila sobre fila 2.

1	0	0
0	1	1
1	0	0

- Operador columna (para cada una de las columnas). Invierte los valores de la columna seleccionada

1	0	0
1	0	0
1	0	0

Aplica operador columna sobre columna 2

1	1	0
1	1	0
1	1	0

- Operador diagonal (sólo aplicable a la diagonal que va desde abajo izquierda a arriba derecha).

1	0	0
1	0	0
1	0	0

Aplica operador diagonal

1	0	1
1	1	0
0	0	0

Importante: si las matrices no son cuadradas, será aplicable y se comenzará desde abajo izquierda

1	0	0
1	0	0
1	0	0
0	0	0

Aplicamos operador diagonal

1	0	0
1	0	1
1	1	0
1	0	0

```
%%======================
%% CÁLCULO DE SUCESORES
%%======================

sucesor(E,S) :- length(E,N),
       (sucesores_filas(E,N,S);sucesores_columnas(E,N,S);opera_diagonal(E,S)).

% Nótese como el operador diagonal no hace falta hacer un "for"

% sucesores operando sobre filas
% es necesaria esta regla para generar todos los candidatos posibles
% (operar sobre fila 1, operar sobre fila 2, ..., operar sobre fila N)
% sería el equivalente a un "for"
sucesores_filas(E,N,S) :- N > 0, N2 is N - 1,
       (opera_fila(E,N2,S);sucesores_filas(E,N2,S)).

% sucesores operando sobre columnas
% de la misma forma que con las filas debemos hacer las columnas
sucesores_columnas(E,N,S) :- N > 0, N2 is N - 1,
       (opera_columna(E,N2,S);sucesores_columnas(E,N2,S)).

%%============
%% OPERADORES
%%============
%% Las reglas de los operadores usados para el cálculo de sucesores, tendrán el
%% siguiente formato (+Estado,+N,-Sucesor) donde N será la fila o columna para
%% los operadores verticales y horizontales.
%% Para el operador de diagonal no hará falta especificar ni fila ni columna
%% (+Estado,-Sucesor)

opera_fila(E,N,S) :- op_filas_aux(E,N,S).
op_filas_aux([H1|T],N,[H1|T2]) :- N > 0, !, N2 is N-1, op_filas_aux(T,N2,T2).
op_filas_aux([H1|T],0,[H2|T]) :- intercambia(H1,1,0,H2).


opera_columna(E,N,S) :- op_columnas_aux(E,N,S).
op_columnas_aux([H1|T1],C,[H2|T2]) :- niega(C,H1,H2),
       op_columnas_aux(T1,C,T2).
op_columnas_aux([],_,[]).

% si sólo admitimos matrices cuadradas
%opera_diagonal(E,F) :- length(E,N), op_diagonal_aux(E,N,S).

% NOTA: para soportar matrices no cuadradas hay que hacer unas comprobaciones
% extra
opera_diagonal([H|T],S) :- length(H,N), length([H|T],F), F == N
       ,op_diagonal_aux([H|T],N,S).
```

```
opera_diagonal([H|T],[H|S]) :- length(H,N), length([H|T],F), F \== N
        ,opera_diagonal(T,S).

op_diagonal_aux([H1|T1],N,[H2|T2]) :- N > 0, N2 is N - 1, niega(N2,H1,H2),
        op_diagonal_aux(T1,N2,T2).
op_diagonal_aux([],_,[]).
```

Comentarios finales

El trabajo a continuar tras éste sería la elaboración de una heurística y su posterior implementación en un A* ya que dependiendo de la configuración para un tablero de 5x5 (o mayor) el tiempo puede dispararse (más si no tiene solución).

Más puzzles (configuraciones iniciales) aquí:
http://www.gameboomers.com/wtcheats/pcTt/ToTheMoon/tothemoon.htm