Each board style is defined using the same predicates and they are located in separate files (*english_board.lp, european_board.lp, german_board.lp, pinwheel_board.lp, simple_3hole_board.lp, simple_4hole_board.lp, simple_5hole_board.lp*). These boards only support 4 directions of movement: up, down, left, right.

In these files the size is defined using the predicate `size(X,Y)`. The german, european and english boards allow for variable sizes as long as they satisfy the board's own unique constraints. A starting position is defined as any point on the board `start(X,Y)`, this is where the first empty hole will be located. Each board uses its own instance of the omitted predicate `(X,Y)` to specify the coordinates that won't be included on the board. Lastly, each board style defines the number of vertices on the board `num_verts(N)`.

In *peg_solitare.lp* the maximum number of moves for any board is defined as:

```
max_moves(N-2) :- num_verts(N).
```

This rule is true for any peg solitaire board that only supports 4 directions of movement. Equivalently we have the same number of time steps:

```
time_steps(0..T) :- max_moves(T).
```

The axis of the board is defined as

```
xdimension(1..X) :- size(X,Y).
ydimension(1..Y) :- size(X,Y).
```

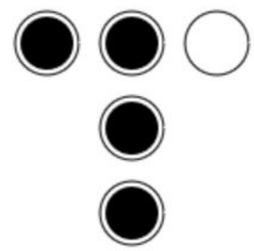
Furthermore, at each time step a move takes place causing vertices on the board change. Hence we define vertices using the predicate `vertex(cell(X,Y), O, T)`. The coordinates are defined by the nested `cell` predicate. `O` takes on the value of 1 when the vertex is occupied by a peg at time step `T` and 0 otherwise. For all coordinates defined by our axis, a vertex gets instantiated if it has not been omitted by the `omitted` predicate. The initial vertices are defined at time step 0 and occupancy is set to 0 if the coordinate is the starting position `start(X,Y)`, and 1 otherwise.

<table>
<tr><td>simple_3hole_board.lp</td><td></td></tr>
<tr><td><pre>%SIMPLE MINIMUM 3 HOLES
board(simple_3hole).
size(3,3).
omitted(1..3,1).
omitted(1..3,3).
start(1,2).
num_verts(3).</pre></td><td></td></tr>
<tr><td>simple_4hole_board.lp</td><td></td></tr>
</table>

```
%SIMPLE 4 HOLES
board(simple_4hole).
size(4,4).
omitted(1..4,1).
omitted(1..4,3).
omitted(1..4,4).
start(2,2).
num_verts(4).
```

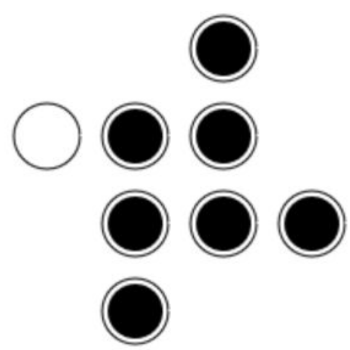
simple_5hole_board.lp

```
%SIMPLE 5 HOLES
board(simple_5hole).
size(3,3).
omitted(1,2..3).
omitted(3,2..3).
start(3,1).
num_verts(5).
```



pinwheel_board.lp

```
%PINWHEEL 8 HOLES
board(pinwheel).
size(4,4).
omitted(1,1).
omitted(1,3..4).
omitted(2,1).
omitted(3,4).
omitted(4,1..2).
omitted(4,4).
start(1,2).
num_verts(8).
```



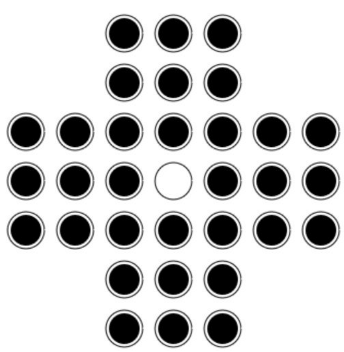
english_board.lp

```
%ENGLISH BOARD
board(english).
%Size-1 must be divisible by 3
size(7,7).
```

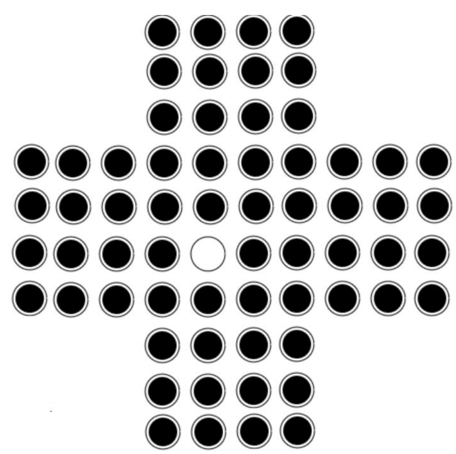
```
:- size(X,X), X < 7.
:- size(X,X), (X-1)\3!=0.
%choose starting position
start(4,4).
%omit blocks in the corner
block1((X-1)/3) :- size(X,Y).
%omitted blocks are block1 x block1 in size
omitted(1..B1, 1..B1) :- size(X,Y), block1(B1).
omitted(1..B1, (Y-B1+1)..Y) :- size(X,Y), block1(B1).
omitted((X-B1+1)..X, 1..B1) :- size(X,Y), block1(B1).
omitted((X-B1+1)..X, (Y-B1+1)..Y) :- size(X,Y), block1(B1).

num_verts(2*B1*B2 + X*B2) :- size(X,Y), block1(B1), B2 = X-2*B1.
```

7 x 7 board



10 x 10 board



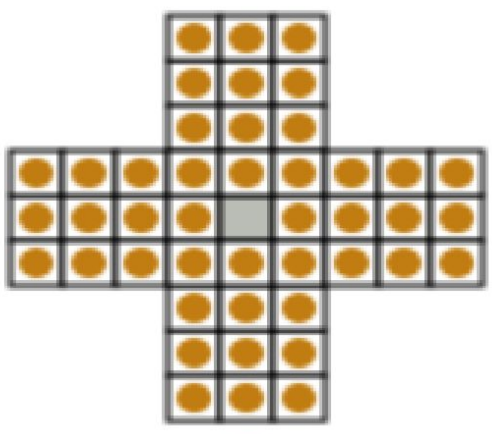
german_board.lp

```
%GERMAN BOARD
%the size of this board is variable
board(german).
%choose a size greater greater than or equal 9x9 that is a multiple of 3
size(9,9).
:- size(X,X), X < 9.
:- size(X,X), X\3!=0.
%choose a starting position
```

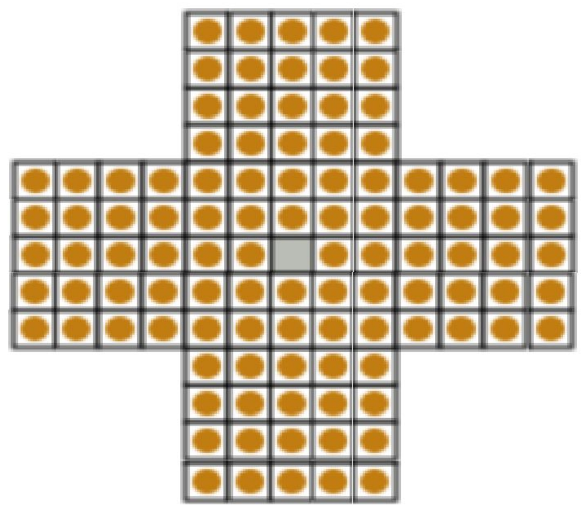
```
start(5,5).
%omit 4 equal size blocks at each corner
block1(X/3) :- size(X,X).
block2(X-B1+1) :- size(X,X), block1(B1).
omitted(1..B1, 1..B1) :- size(X,X), block1(B1).
omitted(1..B1, B2..Y) :- size(X,X), block1(B1), block2(B2).
omitted(B2..X, 1..B1) :- size(X,X), block1(B1), block2(B2).
omitted(B2..X, B2..Y) :- size(X,X), block1(B1), block2(B2).

num_verts(5*B*B) :- size(X,X), B=X/3.
```

9 x 9 board



12 x 12 board



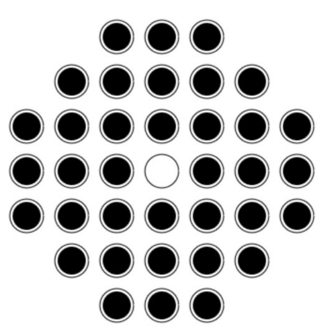
european_board.lp

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%EUROPEAN BOARD
board(european).
%choose either 7x7, 9x9 or 11x11
size(7,7).
%choose a starting position
start(4,4).
%REMOVE CORNERS
corner_depth((N-1)/2) :- size(N,N).
```

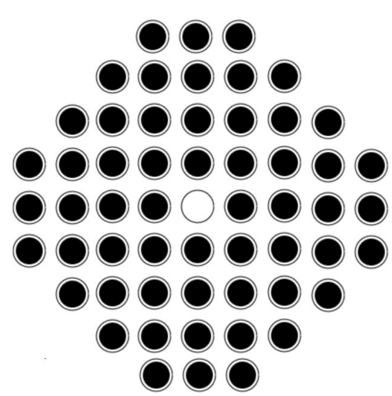
```
corner_cells(1..D, 1..D) :-  size(N,N), corner_depth(D).
%remove left top corner
D*D {omitted(X,1+Y-X) : corner_cells(X,Y)} D*D :- corner_depth(D).
%remove left bottom corner
D*D {omitted(X,D+Y+X) : corner_cells(X,Y)} D*D :- corner_depth(D).
%remove right top corner
D*D {omitted(X+D+1,1-Y+X) : corner_cells(X,Y)} D*D :- corner_depth(D).
%remove right bottom corner
D*D {omitted(X+D+1,N+Y-X) : corner_cells(X,Y)} D*D :- corner_depth(D),
size(N,N).
%remove middle edge cells
omitted(D+1,1) :- corner_depth(D), size(N,N).
omitted(D+1,N) :- corner_depth(D), size(N,N).
omitted(N,D+1) :- corner_depth(D), size(N,N).
omitted(1,D+1) :- corner_depth(D), size(N,N).

num_verts(21) :- size(7,7).
num_verts(37) :- size(9,9).
num_verts(58) :- size(11,11).
```

7x7 board



9x9 board



peg_solitare.lp attempts to solve a peg solitaire board.
At each time step, all possible moves are represented by the predicate:

`possible_move(cell(X2,Y2),cell(X1,Y1),cell(X0,Y0),T)`

Where the peg at **`(X2,Y2)`** jumps over the peg at **`(X1,Y1)`** to get to the empty vertex at **`(X0,Y0)`**.
At each time step all possible moves are considered and only one is chosen. The chosen move is represented by the predicate:

`moves(cell(X2,Y2),cell(X1,Y1),cell(X0,Y0),T)`

Since one move takes place at each time step **`T,`** all the vertices need to be redefined at **`T+1.`** If a vertex was involved in a move, then we update their peg occupancy accordingly, otherwise we don't change their occupancy.

Lastly, we define the predicate **`occupied(N,T)`** to count the number of occupied vertices at each time step. Hence, the following constraint acts as our target:

`:- max_moves(T), not occupied(1,T).`

To see the solution we can show the **`moves`** predicate. My asp program is only able to solve pinwheel_board, simple_3hole_board, simple_4hole_board and simple_5hole_board in a timely manner.

peg_solitaire.lp

```
%By Jonathan Williams (z5162987)
style(simple_3hole).
style(simple_4hole).
style(simple_5hole).
style(pinwheel).
style(european).
style(german).
style(english).

%INITIATE BOARD
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
max_moves(N-2) :- num_verts(N).
time_steps(0..T) :- max_moves(T).

%define vertex cells
xdimension(1..X) :- size(X,Y).
ydimension(1..Y) :- size(X,Y).

%define vertices at time 0
%vertices are cells that are not omitted
vertex(cell(X,Y), 1, 0) :- xdimension(X), ydimension(Y), not omitted(X,Y), not 
start(X,Y).
vertex(cell(Xs,Ys), 0, 0) :- xdimension(Xs), ydimension(Ys), not omitted(Xs,Ys), 
start(Xs,Ys).




%DEFINE MOVES
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%define all possible moves per timestep -- (X2,Y2) jumps over (X1,Y1) to get to
(X0,Y0)
% Right: X2, X1, X0 --> 0, 0, X2
possible_move(cell(X2,Y2),cell(X1,Y1),cell(X0,Y0),T) :-
vertex(cell(X2,Y2),1,T),vertex(cell(X1,Y1),1,T), vertex(cell(X0,Y0),0,T),
X2+1=X1,X1+1=X0,Y2=Y1,Y1=Y0.

% Left: X0, X1, X2, --> X2, 0, 0
possible_move(cell(X2,Y2),cell(X1,Y1),cell(X0,Y0),T) :-
vertex(cell(X2,Y2),1,T),vertex(cell(X1,Y1),1,T), vertex(cell(X0,Y0),0,T),
X2-1=X1,X1-1=X0,Y2=Y1,Y1=Y0.

% Up: Y0/Y1/Y2 --> Y2/0/0
possible_move(cell(X2,Y2),cell(X1,Y1),cell(X0,Y0),T) :-
vertex(cell(X2,Y2),1,T),vertex(cell(X1,Y1),1,T), vertex(cell(X0,Y0),0,T),
Y2-1=Y1,Y1-1=Y0, X2=X1,X1=X0.

% Down: Y2/Y1/Y0 --> 0/0/Y2
possible_move(cell(X2,Y2),cell(X1,Y1),cell(X0,Y0),T) :-
vertex(cell(X2,Y2),1,T),vertex(cell(X1,Y1),1,T), vertex(cell(X0,Y0),0,T),
Y2+1=Y1,Y1+1=Y0, X2=X1,X1=X0.

%MAKE MOVE
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%choose one move per time step from possible moves (don't generate a move on the
final time step)
1{moves(cell(X2,Y2),cell(X1,Y1),cell(X0,Y0),T) :
possible_move(cell(X2,Y2),cell(X1,Y1),cell(X0,Y0),T)}1 :- time_steps(T),
max_moves(M), T<M.

%UPDATE VERTICES EVERY TIMESTEP
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%keep vertices the same each time step if they aren't involved in any moves in the
prev timestep
vertex(cell(X,Y),O,T+1) :- time_steps(T),vertex(cell(X,Y),O,T), not
moves(cell(X,Y),_,_,T),not moves(_,cell(X,Y),_,T), not moves(_,_,cell(X,Y),T).
%update all vertices involved in the move
vertex(cell(X2,Y2),0,T+1) :- moves(cell(X2,Y2),_,_,T).
```

```
vertex(cell(X1,Y1),0,T+1) :- moves(_,cell(X1,Y1),_,T).
vertex(cell(X0,Y0),1,T+1) :- moves(_,_,cell(X0,Y0),T).

%TARGET
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%count number of occupied vertices on the board at each time step
occupied(N,T) :- time_steps(T), #count {cell(X,Y),T: vertex(cell(X,Y),1,T)} = N.
%have 1 occupied vertex on the final timestep
:- max_moves(T), not occupied(1,T).

#show moves/4.
```

Output for **clingo --models 0 peg_solitare.lp pinwheel_board.lp**

```
Solving...
Answer: 1
moves(cell(3,2),cell(2,2),cell(1,2),0) moves(cell(2,4),cell(2,3),cell(2,2),1)
moves(cell(1,2),cell(2,2),cell(3,2),2) moves(cell(4,3),cell(3,3),cell(2,3),3)
moves(cell(2,3),cell(3,3),cell(4,3),5) moves(cell(3,1),cell(3,2),cell(3,3),4)
Answer: 2
moves(cell(3,2),cell(2,2),cell(1,2),0) moves(cell(2,4),cell(2,3),cell(2,2),1)
moves(cell(1,2),cell(2,2),cell(3,2),3) moves(cell(4,3),cell(3,3),cell(2,3),2)
moves(cell(2,3),cell(3,3),cell(4,3),5) moves(cell(3,1),cell(3,2),cell(3,3),4)
SATISFIABLE

Models       : 2
Calls        : 1
Time         : 0.018s (Solving: 0.01s 1st Model: 0.00s Unsat: 0.00s)
CPU Time     : 0.016s
```