```prolog
%%% Equipo 8.
%%% Conformado por:
%%% José Luis Jiménez 10-10839

%%% Proyecto 2 del trimestre SEP-DIC 2013.

:- dynamic jugador1/1.
:- dynamic jugador1/2.
:- dynamic jugador2/1.
:- dynamic jugador2/2.
:- dynamic turno/1.
:- dynamic ficha/3.
:- dynamic ganador/1.

jugador1(persona).
jugador1(peon,'<').
jugador1(rey, '<<').

jugador2(peon, '>').
jugador2(rey, '>>').

peon('>').
peon('<').

rey('>>').
rey('<<').

turno(jugador1).

%%%%%%%%%%%%%%%%%%%%%%                  TABLERO             %%%%%%%%%%%%%%%%%%%%

ficha(1, 2, '<').
ficha(1, 4, '<').
ficha(1, 6, '<').
ficha(1, 8, '<').

ficha(2, 1, '<').
ficha(2, 3, '<').
ficha(2, 5, '<').
ficha(2, 7, '<').

ficha(3, 2, '<').
ficha(3, 4, '<').
ficha(3, 6, '<').
ficha(3, 8, '<').


ficha(6, 1, '>').
ficha(6, 3, '>').
ficha(6, 5, '>').
ficha(6, 7, '>').

ficha(7, 2, '>').
ficha(7, 4, '>').
ficha(7, 6, '>').
ficha(7, 8, '>').

ficha(8, 1, '>').
ficha(8, 3, '>').
ficha(8, 5, '>').
ficha(8, 7, '>').

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%% Predicado para iniciar el juego.
jugar :- hacerPregunta, imprimirTablero, write('\n Turno del jugador 1 \n'), !.
```

```
%%% Predicado para hacer el cambio de turno y verificar si hay un ganador.
actualizarTurno :-  turno(T), jugadorContrario(T,C), retract(turno(T)), assert(turno(C)),
                            not(juegoTerminado), imprimirTurno, jugarMaquina.

%%% Cuando el predicado anterior falla, significa que hay un ganador.
actualizarTurno :-  turno(T), jugadorContrario(T,C), hayGanador(C), !.

%%% Predicados para imprimir por pantalla a que jugador ñe toca jugar.
imprimirTurno :- turno(jugador1) , write('\n Turno del jugador 1 \n').
imprimirTurno :- turno(jugador2) , write('\n Turno del jugador 2 \n').

%%% Predicado para hacer un movimiento de ficha.
jugada(X1,Y1,X2,Y2) :-  X1 =\= X2 , Y1 =\= Y2 , posicionValida(X1,Y1), posicionValida(X2,Y2),
                        ficha(X1,Y1,Z), turno(W), apply(W, [F,Z]), moverFicha(X1,Y1,X2,Y2,F),
                        imprimirTablero, actualizarTurno, !.

%%% Cuando el predicado anterior falla, significa que el movimiento era invalido.
jugada(_,_,_,_)     :-  write('\n Movimiento Invalido \n'), imprimirTablero, !.

%%% Predicado para dirigir la ficha hacia el sitio que debe comer cuando
%%% es necesario comer doble.
repetirJugada(X,Y)  :-  Y2 is Y + 1, X2 is X + 1, comerDeNuevo(X,Y,X2,Y2).
repetirJugada(X,Y)  :-  Y2 is Y - 1, X2 is X + 1, comerDeNuevo(X,Y,X2,Y2).
repetirJugada(X,Y)  :-  Y2 is Y + 1, X2 is X - 1, comerDeNuevo(X,Y,X2,Y2).
repetirJugada(X,Y)  :-  Y2 is Y - 1, X2 is X - 1, comerDeNuevo(X,Y,X2,Y2).
repetirJugada(_,_)  :-  !.

%%% Predicado que se encarga de verificar que es posible comer de nuevo
%%% cuando una ficha ya comio una vez.
comerDeNuevo(X1,Y1,X2,Y2)   :-   posicionValida(X2,Y2), not(casillaVacia(X2,Y2)),
                                  ficha(X1,Y1,Z), turno(W), apply(W, [F,Z]), moverFicha(X1,Y1,X2,Y2,F).

%%% Predicado encargado de hacer el movimiento de una ficha a partir de una
%%% posicion inicial hacia una posicion final.
moverFicha(X1,Y1,X2,Y2,peon)    :-  casillaVacia(X2,Y2), movimientoValido(X1,Y1,X2,Y2,peon,_),
                                    moverVacio(X1,Y1,X2,Y2).

moverFicha(X1,Y1,X2,Y2,peon)    :-  turno(W), verificarCasilla(X2,Y2,W), movimientoValido
(X1,Y1,X2,Y2,peon,Z),
                                    comerFicha(X1,Y1,X2,Y2,Z).

moverFicha(X1,Y1,X2,Y2,rey) :-  casillaVacia(X2,Y2), movimientoValido(X1,Y1,X2,Y2,rey,_),
                                moverVacio(X1,Y1,X2,Y2).

moverFicha(X1,Y1,X2,Y2,rey) :-  turno(W), verificarCasilla(X2,Y2,W), movimientoValido(X1,Y1,X2,Y2,rey,Z),
                                comerFicha(X1,Y1,X2,Y2,Z).

%%% Se usa cuando la posicion X2, Y2 esta vacia.
moverVacio(X1,Y1,X2,Y2) :- buscarFicha(X1,Y1,X2,Y2,F), retract(ficha(X1,Y1,_)), assert(ficha(X2,Y2,F)).

%%% Predicado para verificar que los peones y reyes se mueven en direcciones
%%% validas dentro del tablero. En el caso de los peones, ellos solo se mueven
%%% en ciertas direcciones dependiendo del jugador que juegue actualmente.
movimientoValido(X1,Y1,X2,Y2,peon,derecha)          :-  turno(jugador1),  X2 =:= X1 + 1, Y2 =:= Y1 + 1.
movimientoValido(X1,Y1,X2,Y2,peon,izquierda)        :-  turno(jugador1),  X2 =:= X1 + 1, Y2 =:= Y1 - 1.
movimientoValido(X1,Y1,X2,Y2,peon,antiderecha)      :-  turno(jugador2),  X2 =:= X1 - 1, Y2 =:= Y1 + 1.
movimientoValido(X1,Y1,X2,Y2,peon,antizquierda)     :-  turno(jugador2),  X2 =:= X1 - 1, Y2 =:= Y1 - 1.
movimientoValido(X1,Y1,X2,Y2,rey,derecha)           :-  diagonalDerecha(X1,Y1,X2,Y2).
movimientoValido(X1,Y1,X2,Y2,rey,izquierda)         :-  diagonalIzquierda(X1,Y1,X2,Y2).
movimientoValido(X1,Y1,X2,Y2,rey,antiderecha)       :-  antidiagonalDerecha(X1,Y1,X2,Y2).
movimientoValido(X1,Y1,X2,Y2,rey,antizquierda)      :-  antidiagonalIzquierda(X1,Y1,X2,Y2).

%%% Predicado encargado de eliminar una ficha del tablero. En la posicion
%%% X2, Y2 se encuentra una ficha del jugador contrario al que esta jugando.
comerFicha(X1,Y1,X2,Y2,derecha)         :-  X3 is X2 + 1, Y3 is Y2 + 1, posicionValida(X3,Y3),
                                            casillaVacia(X3,Y3), buscarFicha(X1,Y1,X3,Y3,Z),
                                            retract(ficha(X1,Y1,_)), retract(ficha(X2,Y2,_)),
```

```
                                        assert(ficha(X3,Y3,Z)), repetirJugada(X3,Y3) , !.

comerFicha(X1,Y1,X2,Y2,izquierda)       :-  X3 is X2 + 1, Y3 is Y2 - 1, posicionValida(X3,Y3),
                                            casillaVacia(X3,Y3), buscarFicha(X1,Y1,X3,Y3,Z),
                                            retract(ficha(X1,Y1,_)), retract(ficha(X2,Y2,_)),
                                            assert(ficha(X3,Y3,Z)), repetirJugada(X3,Y3), !.

comerFicha(X1,Y1,X2,Y2,antiderecha)     :-  X3 is X2 - 1, Y3 is Y2 + 1, posicionValida(X3,Y3),
                                            casillaVacia(X3,Y3), buscarFicha(X1,Y1,X3,Y3,Z),
                                            retract(ficha(X1,Y1,_)), retract(ficha(X2,Y2,_)),
                                            assert(ficha(X3,Y3,Z)), repetirJugada(X3,Y3), !.

comerFicha(X1,Y1,X2,Y2,antizquierda)    :-  X3 is X2 - 1, Y3 is Y2 - 1, posicionValida(X3,Y3),
                                            casillaVacia(X3,Y3), buscarFicha(X1,Y1,X3,Y3,Z),
                                            retract(ficha(X1,Y1,_)), retract(ficha(X2,Y2,_)),
                                            assert(ficha(X3,Y3,Z)), repetirJugada(X3,Y3), !.

%%% Predicado que toma una ficha en la posicion X1, Y1 y, si es un rey
%%% retorna el mismo rey. En caso de ser un peon, revisa la posicion
%%% X2 y, en caso de estar en una posicion en la que un peon pasa a rey,
%%% retorna la ficha correspondiente a  un rey; en caso contrario, retorna
%%% la ficha de un peon.
buscarFicha(X1,Y1,_,_,Z)    :- ficha(X1,Y1,Z), rey(Z).
buscarFicha(_,_,X2,_,Z)     :- turno(T), verificarRey(X2,T), apply(T,[rey,Z]).
buscarFicha(_,_,_,_,Z)      :- turno(T), apply(T,[peon, Z]).

%%% Toma el valor de X y dependiendo del turno, tiene exito si es un valor
%%% en que un peon pasa a ser rey.
verificarRey(X,jugador1)    :-  X =:= 8.
verificarRey(X,jugador2)    :-  X =:= 1.

%%% Predicado para verificar que en la posicion X, Y no se encuentra una ficha
%%% del jugador W.
verificarCasilla(X,Y,W) :-  apply(W, [peon, Z1]), not(ficha(X,Y,Z1)), apply(W, [rey, Z2]), not(ficha
(X,Y,Z2)).

%%% Predicado que toma una posicion X,Y y verifica que la posicion X2, Y2
%%% se encuentra ubicada en el camino diagonal inferior derecho. Ademas
%%% verifica que este camino esta libre.
diagonalDerecha(X1,Y1,X2,Y2)          :-  X2 > X1, Y2 > Y1, X3 is X2 - X1, Y3 is Y2 - Y1,
                                          Y3 =:= X3, X4 is X1 + 1, Y4 is Y1 + 1,
                                          caminoLibre(X4,Y4,X2,Y2,derecha).

%%% Predicado que toma una posicion X,Y y verifica que la posicion X2, Y2
%%% se encuentra ubicada en el camino diagonal inferior izquierdo. Ademas
%%% verifica que este camino esta libre.
diagonalIzquierda(X1,Y1,X2,Y2)        :-  X2 > X1, Y1 > Y2, X3 is X2 - X1, Y3 is Y1 - Y2, Y3 =:= X3, X4 is
X1 + 1,
                                          Y4 is Y1 - 1, caminoLibre(X4,Y4,X2,Y2,izquierda).

%%% Predicado que toma una posicion X,Y y verifica que la posicion X2, Y2
%%% se encuentra ubicada en el camino diagonal superior derecho. Ademas
%%% verifica que este camino esta libre.
antidiagonalDerecha(X1,Y1,X2,Y2)      :-    X1 > X2, Y2 > Y1, X3 is X1 - X2, Y3 is Y2 - Y1,
                                            Y3 =:= X3, X4 is X1 - 1, Y4 is Y1 + 1,
                                            caminoLibre(X4,Y4,X2,Y2,antiderecha).

%%% Predicado que toma una posicion X,Y y verifica que la posicion X2, Y2
%%% se encuentra ubicada en el camino diagonal superior izquierdo. Ademas
%%% verifica que este camino esta libre.
antidiagonalIzquierda(X1,Y1,X2,Y2)    :-    X1 > X2, Y1 > Y2, X3 is X1 - X2, Y3 is Y1 - Y2,
                                            Y3 =:= X3, X4 is X1 - 1,  Y4 is Y1 - 1,
                                            caminoLibre(X4,Y4,X2,Y2,antizquierda).

%%% Predicado para verificar que el camino marcado por X,Y -> X2,Y2 esta libre.
caminoLibre(X,Y,X,Y,_)                  :-  !.
caminoLibre(X1,Y1,X2,Y2,derecha)        :-  casillaVacia(X1,Y1), X3 is X1 + 1, Y3 is Y1 + 1,
```

```
                                        caminoLibre(X3,Y3,X2,Y2,derecha).

caminoLibre(X1,Y1,X2,Y2,izquierda)      :-  casillaVacia(X1,Y1), X3 is X1 + 1, Y3 is Y1 - 1,
                                            caminoLibre(X3,Y3,X2,Y2,izquierda).

caminoLibre(X1,Y1,X2,Y2,antiderecha)    :-  casillaVacia(X1,Y1), X3 is X1 - 1, Y3 is Y1 + 1,
                                            caminoLibre(X3,Y3,X2,Y2,antiderecha).

caminoLibre(X1,Y1,X2,Y2,antizquierda)   :-  casillaVacia(X1,Y1), X3 is X1 - 1, Y3 is Y1 - 1,
                                            caminoLibre(X3,Y3,X2,Y2,antizquierda).

%%% Verifica que una casilla esta vacia.
casillaVacia(X,Y) :- not(ficha(X,Y,_)).

%%% Comprueba que la posicion X,Y esta dentro del tablero.
posicionValida(X,Y) :- (X =< 8), (X >= 1), (Y =< 8), (Y >= 1).

%%% Pregunta al usuario si desea que el jugador 2 sea la maquina.
hacerPregunta :- write('\nDesea jugar contra la maquina? (S,N)\n'), read(X), actualizarJugadores(X).

%%% Introduce dentro de la base de conocimiento de que tipo es el jugador 2.
actualizarJugadores('s')    :-  assert(jugador2(maquina)).
actualizarJugadores('n')    :-  assert(jugador2(persona)).
actualizarJugadores(_)      :-  hacerPregunta.

%%% Comprueba que el juego a terminado verificando si al jugador de turno
%%% todavia le quedan movimientos por hacer.
juegoTerminado  :-  movimientosPosibles(L), length(L,N), N =:=0.

%%% Imprime por pantalla quien es el ganador del juego.
hayGanador(jugador1)    :- write('\n Ha ganado el jugador 1 \n'), !.
hayGanador(jugador2)    :- write('\n Ha ganado el jugador 2 \n'), !.

%%% Recibe una instancia de un jugador y retorna el atomo para el jugador
%%% contrario.
jugadorContrario(jugador1,jugador2) :- !.
jugadorContrario(jugador2,jugador1) :- !.

%%% Recibe dos fichas y retorna si la segunda de estas esta vacia o si el
%%% el jugador de turno puede comer la ficha dentro de esta.
casillaPermitida(_,ficha(X,Y),vacia)    :-  posicionValida(X,Y), casillaVacia(X,Y).
casillaPermitida(An,ficha(X,Y),comer)   :-  posicionValida(X,Y), verificarCasilla(X,Y,jugador2),
                                            verificarSiguiente(An,ficha(X,Y)).

%%% Verifica que la posicion siguiente a X2, Y2 es valida y esta vacia para que
%%% el jugador de turno puede comer en X2, Y2 y saltar.
verificarSiguiente(ficha(X1,Y1) , ficha(X2,Y2)) :-  X2 > X1, Y2 > Y1, X3 is X2 + 1 ,
                                                    Y3 is Y2 + 1, posicionValida(X3,Y3),
                                                    casillaVacia(X3,Y3).

verificarSiguiente(ficha(X1,Y1) , ficha(X2,Y2)) :-  X2 < X1, Y2 > Y1, X3 is X2 - 1 ,
                                                    Y3 is Y2 + 1, posicionValida(X3,Y3),
                                                    casillaVacia(X3,Y3).

verificarSiguiente(ficha(X1,Y1) , ficha(X2,Y2)) :-  X2 > X1, Y2 < Y1, X3 is X2 + 1 ,
                                                    Y3 is Y2 - 1, posicionValida(X3,Y3),
                                                    casillaVacia(X3,Y3).

verificarSiguiente(ficha(X1,Y1) , ficha(X2,Y2)) :-  X2 < X1, Y2 < Y1, X3 is X2 - 1 ,
                                                    Y3 is Y2 - 1, posicionValida(X3,Y3),
                                                    casillaVacia(X3,Y3).

%%% Si el movimiento de la ficha An a la ficha Ac es valido, inserta un
%%% par(ficha(X,Y),ficha(X2,Y2)) dentro de la lista Z.
agregarFicha(An,Ac,Z,R)     :- casillaPermitida(An,Ac,P), append(Z,[par(An,Ac,P)],R).
agregarFicha(_,_,Acc,Acc)   :- !.
```

```
%%% Retorna una lista con dos pares que espicifican las dos posiciones a las
%%% que puede moverse el peon que esta en la posicion X,Y.
movimientoPosible(ficha(X,Y,'>'),Z) :-  X1 is X - 1, Y1 is Y - 1,
                                        X2 is X - 1, Y2 is Y + 1,
                                        Z = [
                                                par(ficha(X,Y),ficha(X1,Y1)),
                                                par(ficha(X,Y),ficha(X2,Y2))
                                            ].

%%% Retorna una lista con todos los pares posibles a los que puede moverse
%%% el rey que esta en la posicion X,Y.
movimientoPosible(ficha(X,Y,'>>'),Z)    :-  agregarDerecha(X,Y,X,Y,[],R1),
                                            agregarIzquierda(X,Y,X,Y,R1,R2),
                                            agregarAntiderecha(X,Y,X,Y,R2,R3),
                                            agregarAntizquierda(X,Y,X,Y,R3,Z).

%%% Los cuatro predicados que vienen a continuacion, sirven para agregar
%%% todas las posiciones posibles a las que puede moverse un rey. Estos van
%%% verificando posicion por posicion si la casilla esta libre hasta llegar al
%%% fondo del tablero o encontrar otra ficha.
agregarDerecha(X0,Y0,X,Y,E,S)       :-  X1 is X + 1, Y1 is Y + 1,
                                        continuar(X0,Y0,X1,Y1,E,Aux,agregarDerecha),
                                        agregarLista(X0,Y0,X1,Y1,Aux,S), !.

agregarIzquierda(X0,Y0,X,Y,E,S)     :-  X1 is X + 1, Y1 is Y - 1,
                                        continuar(X0,Y0,X1,Y1,E,Aux, agregarIzquierda),
                                        agregarLista(X0,Y0,X1,Y1,Aux,S), !.

agregarAntiderecha(X0,Y0,X,Y,E,S)   :-  X1 is X - 1, Y1 is Y + 1,
                                        continuar(X0,Y0,X1,Y1,E,Aux, agregarAntiderecha),
                                        agregarLista(X0,Y0,X1,Y1,Aux,S), !.

agregarAntizquierda(X0,Y0,X,Y,E,S)  :-  X1 is X - 1, Y1 is Y - 1,
                                        continuar(X0,Y0,X1,Y1,E,Aux, agregarAntizquierda),
                                        agregarLista(X0,Y0,X1,Y1,Aux,S), !.

%%% Si la posicion X1, Y1 es valida (esta dentro del tablero y esta libre o
%%% contiene una ficha del contrario), agregar el par(ficha(X0,Y0),ficha(X1,Y1))
%%% a la lista L y la retorna en S.
agregarLista(X0,Y0,X1,Y1,L,S)   :-  posicionValida(X1,Y1), append(L,[par(ficha(X0,Y0),ficha(X1,Y1))],S).
agregarLista(_,_,_,_,L,L)       :-  !.

%%% Verifica que la posicion X, Y esta libre para seguir buscando en la direccion
%%% indicada por F. Usada para buscar las posiciones a las que pueden moverse
%%% los reyes.
continuar(X0,Y0,X,Y,E,S,F)  :- posicionValida(X,Y), casillaVacia(X,Y), apply(F, [X0,Y0,X,Y,E,S]).
continuar(_,_,_,_,E,E,_)    :- !.

%%% Recibe una lista de todos los movimientos posibles que puede hacer un jugador
%%% en un turno. El predicado se encarga se filtrar estos movimientos y quedarse
%%% con aquellos que de verdad resultan en un movimiento valido.
verificarMovimientos([],Acc,Acc)                :-  !.
verificarMovimientos([par(An,Ac) | Tail],Acc,R) :-  agregarFicha(An,Ac, Acc, RAcc),
                                                    verificarMovimientos(Tail, RAcc, R).

%%% Toma una lista con todas las fichas dentro del tablero y retorna todos los
%%% movimientos que pueden hacer estas fichas si no existieran fichas
%%% contrarias.
listaMovimientos([],R,R)            :-  !.
listaMovimientos([X | Tail],Z,S)    :-  apply(movimientoPosible,[X,L]), verificarMovimientos(L,[],R),
                                        append(Z,R,Aux), listaMovimientos(Tail, Aux,S), !.

%%% Toma una lista y retorna un elemento random de esta.
seleccionarRandom(L,S)    :-  random(R), length(L,N), F is 1/N, seleccionarElemento(L,R,F,S).

%%% Predicado auxiliar de seleccionar el elemento random de la lista.
seleccionarElemento([X | _ ], R, F,X)   :-  R < F.
```

```
seleccionarElemento([_ | Tail], R, F,S) :-  R >= F, N is F + F, seleccionarElemento(Tail, R, N, S).

%%% Predicado para separar un par en la posiciones de la fichas fichas
%%% que lo contienen y aplicar la jugada correspondiente.
aplicarJugada(par(ficha(X1,Y1),ficha(X2,Y2))) :- jugada(X1,Y1,X2,Y2).

%%% Se encarga de definir el orden en que se van a aplicar los predicados
%%% correspondientes al movimiento de la maquina.
%%% 1) Se buscan todos los movimientos posibles que puede hacer el jugador de turno.
%%% 2) Separa los movimientos en aquellos en los que el jugador puede comer y los que no.
%%% 3) Intenta seleccionar un elemento de la lista de los movimientos en los
%%% que puede comer. En caso de no hallar alguno, busca en la lista de
%%% movimientos en los que no puede comer.
%%% 4) Realiza la jugada seleccionada.
moverMaquina    :-  movimientosPosibles(R), separarPorComer(R,[],[],V,C),
                    seleccionarJugada(C,V,P) , aplicarJugada(P), !.

%%% Predicado para seleccionar primero una jugada de los movimientos
%%% en los que se puede comer. Si no encuentra, selecciona un movimiento
%%% de la lista de los que no puede comer.
seleccionarJugada([],V,S) :- seleccionarRandom(V,S), !.
seleccionarJugada(C,_,S)  :- seleccionarRandom(C,S), !.

%%% Toma una lista de judadas y la separa en dos listas. En una quedan los
%%% movimientos en los que el jugador puede comer, y en la ontra en los que no.
separarPorComer([], AccV,AccC,AccV,AccC)    :-  !.

separarPorComer([par(ficha(X,Y), ficha(X1,Y1), comer) | Tail],AccV,AccC,SV,SC)
                                            :-  append(AccC,[par(ficha(X,Y), ficha(X1,Y1))],Aux),
                                                separarPorComer(Tail,AccV,Aux,SV,SC), !.

separarPorComer([par(ficha(X,Y), ficha(X1,Y1), vacia) | Tail],AccV,AccC,SV,SC)
                                            :-  append(AccV,[par(ficha(X,Y), ficha(X1,Y1))],Aux),
                                                separarPorComer(Tail,Aux,AccC,SV,SC), !.

%%% Predicado que verifica si el jugador dos es una maquina y hace la jugada
%%% automaticamente.
jugarMaquina    :-  jugador2(maquina), moverMaquina, !.
jugarMaquina    :-  !.

%%% Retorna todos los movimientos posibles del jugador de turno.
movimientosPosibles(R)  :-  turno(T),
                            apply(T,[peon,Peon]), findall(ficha(X,Y,Peon),ficha(X,Y,Peon),L1),
                            apply(T,[rey,Rey]), findall(ficha(X,Y,Rey),ficha(X,Y,Rey),L2),
                            append(L1, L2, L), listaMovimientos(L,[],R),!.

%%% Todos los predicados siguiente se encargan de imprimir la lista por pantalla.
imprimirTablero    :-   write('\n'),
                        write('     1    2    3    4    5    6    7    8  \n'),
                        imprimirFilas(1),
                        !.

imprimirFilas(9).
imprimirFilas(X) :- write(X), write('  |  '), imprimirFila(X), write('\n'), X1 is X + 1, imprimirFilas
(X1).

imprimirFila(X) :- imprimirLista(X,1).

imprimirLista(_,9).
imprimirLista(F, C) :- ficha(F,C,Z), peon(Z), write(Z), write(' |  '), C1 is C + 1, imprimirLista(F,C1).
imprimirLista(F, C) :- ficha(F,C,Z), rey(Z), write(Z), write('|  '), C1 is C + 1, imprimirLista(F,C1).

imprimirLista(F, C) :- write('  |  ') , C1 is C + 1, imprimirLista(F,C1).
```