N-Queens.

Daniel Camba Lamas, Diego Casanova José y Román Puga Quintairos.

1) Estrategia

En primer lugar se obtiene de la **BC** todos los *percepts* de `queen(X,Y)` haciendo con ellos una lista a partir de la cual se calculan las *posiciones libres*, es decir que no están amenazadas en base a las reinas colocadas actualmente. A mayores se obtienen todos los *percepts* de `block(X,Y)`; para seguir, a la previa lista de *posiciones libres*, le quitamos aquellas que están bloqueadas para obtener la realmente lista de *posiciones libres* evaluando tanto amenzas como bloqueos. Y es en base a ésta que se escoge una posición al azar, para colocar una Reina o un Bloqueo, según si el agente es Blancas/Negras o Bloqueador.

A mayores en el instante inmediatamente anterior a la colocación de una reina o un bloqueo se comprueba que de forma paralela otro agente no haya colocado un elemento en la casilla que acabamos de calcular para colocar.

Motivación

Recopilar la totalidad de casillas libres, da un abanico de opciones mucho mayor que una comprobación lineal o similares, además que podría permitir la inclusión de estadísticas para la posterior selección de la posición en la que posicionar la reina o el bloqueador. En esta ocasión, dudamos entre escoger la última posición de la lista ó una aleatoria, finalmente se ha utilizado la selección aleatoria ya que aunque resulta menos controlable para nosotros, también es menos predecible para el enemigo.

2) Implementación

La implementación de los agentes se divide en hechos y reglas en la base de conocimientos y objetivos y tareas en jason para el propio agente. En nuestra base de conocimientos hay 9 predicados prolog que se encargan de la obtención de las casillas no amenzadas.

mas2j

En esta segunda entrega se han definido alias en base al rol que juega cada agente, y añadiendo en esta parte el turno en el que les corresponde jugar, siendo el bloqueador capaz de jugar en ambos turnos.

```
MAS mars {
    infrastructure: Centralised
    environment: QueensEnv
    agents:
        block r2 [beliefs="playAs(0),playAs(1)"];
        white r0 [beliefs="playAs(0)"];
        black r1 [beliefs="playAs(1)"];
}
```

Diferencias entre `r0.asl` y `r1.asl`

En esta esegunda entrega la diferencia entre los agentes que gestionan blancas y negras se limitan simplemente a que el agente que gestiona las fichas blancas *r0* es quien inicia la partida, forzando el percept de player. A mayores, se eliminata dicho percept tras dar comienzo el juego para evitar confilctos con las actualizaciones del entorno ya que al ser este percept "forzado" de origen ***self*** puede que el entorno no sea capaz de eliminarlo.

```
/* JASON */
!start.
+!start: true <- +player(0); -player(0).
```

Predicados

iterator(X,Y,L,R): X e Y nos dan el tamaño del tablero, L el estado actual del tablero y R las casillas no amenazadas.

link(X,Y,R): Un simple metodo para concatenar dos listas en una *R*esultante.

lookfor(X,Y,L,R,S): Realiza la comprobación de una hilera de casiilas, X la primera posicion normalmente 0, Y será la posicion a comprobar , L el estado del tablero, R las casillas no atacadas y S el tamaño de tablero.

threat(X,L,S): X es la posición a comprobar, L las lista con la que tenemos que comparar y de la cual cogemos la primera reina, S es el tamaño del tablero. Comprobamos verticales, horizonatales y, diagonales mediante la llamada a `checkloop` que nos realiza la comprobación de la totalidad de las diagonales, mediante `check` para comprobar la diagonal a una distancia P y `onemore` para aumentar dicha P, osea para aumentar el alcance.

parser([[X,Y]|_],X,Y): Nos permite coger las coordenadas de la primera posición de una lista.

```
/* PROLOG */

iterator(Y,Y,L,R):- lookfor(Y,Y,L,R,Y).
iterator(X,Y,L,K):- X1=X+1 & link(MF,R,K) &
lookfor(X,Y,L,MF,Y) & iterator(X1,Y,L,R).

link([], Cs, Cs).
link([A|As],Bs,[A|Cs]):- link(As, Bs, Cs).

lookfor(X,0,L,[],S):- threat([X,0],L,S).
lookfor(X,0,_,[[X,0]],_).
lookfor(X,Y,L,MF,S):- Y1=Y-1 & threat([X,Y],L,S) &
lookfor(X,Y1,L,MF,S).
lookfor(X,Y,L,[[X,Y]|MF],S):- Y1=Y-1 & lookfor(X,Y1,L,MF,S).

threat([],[],_).
threat([X,_],[[X,_]|_],_).
threat([_,Y],[[_,Y]|_],_).
threat(Q,[Car|Cdr],P):- threat(Q,Cdr,P) | checkloop(Q,P,
[Car|Cdr]).

checkloop([X1,X2],P,[[C1,C2]|R]):-
check(X1,X2,P,[[C1,C2]|R])|onemore([X1,X2],P,[[C1,C2]|R]).

onemore([X1,X2],P,[[C1,C2]|R]):-
P>0 & P1=P-1 & checkloop([X1,X2],P1,[[C1,C2]|R]).

check(X1,X2,P,[[C1,C2]|R]):- X1=C1+P & X2=C2+P | X1=C1-P &
X2=C2+P | X1=C1+P & X2=C2-P | X1=C1-P & X2=C2-P | X1==C1 &
X2==C2.

parser([[X,Y]|_],X,Y).
```

Gestión de turno.

En caso de que el jugador indicado por el entorno (*player()*), coincida con nuestro identificador (*playAS()*), iniciaremos el proceso de *Cálculo-Comprobación-Colocación* para una reina o bloque.

```
+player(P) : playAs(P) <- !do.
```

Plan `+!do`

```
+!do: true <-
    !getPos(X,Y); //Encontrará una posición libre para colocar un elemento.
    !check(X,Y). //Forzará una comprobación en el instante previo a colocar.
```

Plan `+!getPos(X,Y)`

```
+!getPos(X,Y) <-
    ?size(ES); S=ES-1; //Tamaño del tablero con el que vamos trabajar.
    .findall([I,J],queen(I,J),ALL); //Recoge todas las reinas colocadas.
    ?iterator(0,S,ALL,FL); //Calcula las posiciones no amenazadas.
    .findall([N,M],block(N,M),BL); //Recoge todas las casillas bloqueadas.
    .difference(FL,BL,RL); //Calcula libres en base a amenazas y bloqueos.
    if(not(.empty(RL))) //Si hay casillas libres...
    {
      .shuffle(RL,SRL); //Desordenamos la lista de casillas libres.
      ?parser(SRL,X,Y); //Escogemos el primer par de la lista.
    }
    else //Si no hay casillas libres...
    {
      .my_name(ME); //Obtenemos el nombre del agente.
      .kill_agent(ME); //Finalizamos su ejecución.
    }.
```

Plan `+!check(X,Y)`

```
+!check(X,Y)<-
    if(block(X,Y) | queen(X,Y)) //Si la posición ha sido ocupada...
    { !do } //Repito el proceso.
    else //Si en el instante anterior a colocar, la posicioón sigue libre
    { queen(X,Y) }. //Coloco una reina. (Ó un bloque en el caso del Bloqueador)
```