PRACTICA 2 SI NREINAS

1 OBJETIVOS

Desarrollar un programa (que funcione en Jason) que intente ganar al juego de colocar N reinas sin amenaza en un tablero de tamaño 2N (siendo N variable) contra otro agente programado por otro grupo siendo proporcionado el entorno.

2 CONDICIONES

1_El programa (por turnos) podrá jugar con blancas o con negras.
2_Si el programa juega con blancas gana si es capaz de colocar N reinas en el tablero sin que se amenacen entre sí ni sean amenazadas por las reinas del oponente, o es capaz de colocar un número de reinas mayor que su oponente.
3_Si el programa juega con negras gana si es capaz de colocar en el tablero al menos tantas reinas como su oponente.
4_Cada jugador no puede colocar una reina en una celda que este cubierta por otra reina (ya sea suya o de su adversario) Cada jugador colocará una reina por turno.
5_Si el programa no verifica las reglas, se considerará que la práctica está suspensa.

3 ESTRATEGIA

La estrategia seleccionada para resolver el problema se puede dividir en dos partes, en la primera el algoritmo a partir de la posición de las reinas desplegadas en el tablero genera una lista con las coordenadas de todas las posiciones que noe stan amenazadas por una reina de esta manera partimos de todas las posibles ubicaciones de la siguiente reina a posicionar dejando a la segunda parte del algoritmo la estrategia de colocación, para esta práctica nos hemos decantado por seleccionar una posicion mediante un metodo random, de manera que se selecciona unas coordenadas aleatorias dentro de las proporcionadas por la lista generada en la primera parte de nuestro algoritmo.

3.1 MOTIVO DE ELECCIÓN ESTRATEGIA

Para la primera parte del algoritmo nos decidimos por recopilar todas las posiciones no amenazadas porque nos pareció que nos daba muchas mas posibilidades para la selección de la estrategia de colocación en la segunda parte del algoritmo, como posible ampliación de esta parte se podría ordenar la lista de manera que las coordenadas que mas posiciones no amenazadas amenacen este de primeras con el fin de dificultar al ajente enemigo.
Para la segunda parte del algoritmo nos decidimos por seleccionar las coordenadas de manera aleatoria debido a falta de tiempo, pero podría mejorarse implementando diferentes estrategias e incluso utilizar la que mas convenga en cada momento.

3.2 PROS Y CONTRAS ESTRATEGIA

3.2.1PROS

a)La discriminacion de las posiciones no amenazadas da mucho juego para la selección de estrategia de colocación
b)Al colocarse en una de estas posiciones te aseguras de que la reina siempre va a estar colocada en una posición viable
c)mucho margen para seleccionar la estrategia de colocación al tener todas las posiciones posibles
d)mucho margen de amplicación y mejora del algoritmo

3.2.1CONTRAS

a)independientemente de la estrategia de colocación siempre se van a comprobar todas las coordenadas para discriminar las no amenazadas
b)no es el algoritmo mas rápido

4 IMPLEMENTACIÓN

La implementación del agente se divide en hechos y reglas en la base de conocimientos y objketivos y tareas en el propio agente en jason

4.1 BASE DE CONOCIMIENTOS

En total en la base de conocimientos hay 21 predicados prolog en los que radica la mayor parte de la logica de

seleccion de coordenadas disponibles y colocación de la reina

PREDICADOS

qfor(X,Y,L,R):donde X e Y son el tamaño del tablero cuadrado, es decir si es un 4x4 X sería 0 y Y seria 3, L la lista con las posiciones de las reinas en el tablero y R la lista con las posiciones no amenazadas

```
qfor(Y,Y,L,R):- mirafila(Y,Y,L,R,Y).

qfor(X,Y,L,K):- X1=X+1 &
                    concat(MF,R,K)&
                    mirafila(X,Y,L,MF,Y) &
                    qfor(X1,Y,L,R).
```

mirafila(X,Y,L,R,S):comprueba una fila siendo X la primera posicion normalmente 0 ,Y la posicion a comprobar , L la lista de posiciones de reinas, R las posiciones no atacadas a devolver y S el tamaño de tablero

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

ataca(X,Y,S):comprueba si una posicion X es atacada en horizontal o vertical por la primera reina de la lista Y siendo S el tamaño del tablero necesario para atacaDiag

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

ataca(Q,[Car|Cdr],P):-  ataca(Q,Cdr,P)|
                        atacaDiag(Q,P,[Car|Cdr]).
```

atacaDiag(X,P,Y):comprueba si la posicion X es amenazada por la primera reina de la lista Y a una distancia P en alguna de sus diagonales.

```
atacaDiag([X1,X2],P,[[C1,C2]|R]):-  comprueba(X1,X2,P,[[C1,C2]|R])|
                                    nextlvl([X1,X2],P,[[C1,C2]|R]).
```

nextlvl(X,P,Y):aumenta la distancia P para comprobar la totalidad de las diagonales

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

comprueba(X1,X2,P,Y):realiza las comprobaciones finales en las diagonales a la distancia P siendo X1 y X2 las coordenadas desglosadas y Y la lsita de reinas.

```
comprueba(X1,X2,P,[[C1,C2]|R]):-comprueba1(X1,X2,P,C1,C2)|
                                comprueba2(X1,X2,P,C1,C2)|
                                comprueba3(X1,X2,P,C1,C2)|
                                comprueba4(X1,X2,P,C1,C2)|
                                comprueba5(X1,X2,P,C1,C2)|
                                comprueba(X1,X2,P,R).


comprueba1(X1,X2,P,C1,C2):- X1=C1+P & X2=C2+P.
```

```
comprueba2(X1,X2,P,C1,C2):- X1=C1-P & X2=C2+P.
comprueba3(X1,X2,P,C1,C2):- X1=C1+P & X2=C2-P.
comprueba4(X1,X2,P,C1,C2):- X1=C1-P & X2=C2-P.
comprueba5(X1,X2,_,C1,C2):- X1=C1 & X2=C2.
```

monta(L,L2):se le pasa una lista de elementos tipo queen(X,Y) y devuelve una lista de pares de objetos [X,Y].

```
monta([],[]).
monta([Car|Cdr],[[X,Y]|L]):- despieza(Car,X,Y) &
                             monta(Cdr,L).
```

despieza(q(X,Y),X,Y):se le pasa un objeto de tiepo queen(X,Y) y devuelve X e Y.

```
despieza([],[],[]).
despieza(q(X,Y),X,Y).
```

concat(X, Y, R):concatena las listas X e Y en R.

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

elemrandom(L,N,E):devuelve E siendo el elemento N de L, en un principio se pretendía conseguir el numero random en prolog y funcionaba pero daba problemas en jason asi que incluimos N que es generada en Jason (por eso esta comentado rdn)

```
elemrandom(L,N,E):-//rdn(L,N)&
                get(L,N,E).
```

get(X,Y,E):devuelve el elemento E que es el numero Y de la lista X.

```
get([Car|Cdr],0,Car).
get([Car|Cdr],N,X):- N1=N-1 &
            get(Cdr,N1,X).
```

longitud(L,Lon):devuelve la longitud de la lista L.

```
longitud([],0).
longitud([_|T],N):-longitud(T,N0) & N=N0 + 1.
```

parset([X,Y],X,Y):dado un par de elementos devuelve los elementos.

```
parset([X,Y],X,Y).
```

checkTurno(Jugador,Turno,X):comprueba si es el turno del jugador X.

```
checkTurno(Jugador,Turno,1):- Jugador = Turno.
checkTurno(Jugador,Turno,0):- not(Jugador=Turno).
```

turno([Car|Cdr],Car):devuelve el primer elemento de una lista.

```
turno([Car|Cdr],Car).
```

4.2 IMPLEMENTACIÓN AGENTE