Práctica 3

Esteban Omelio Puentes Silveira
virtualevan@gmail.com, Xosé Ramón Fernández Oxea #3 1ºG

FORMACIÓN ACADÉMICA
2008-2011 Título de Bachiller (Homologado). IPVCE Máximo Gómez Báez (Camagüey-Cuba)
2011-2013 FP2/ Técnico en Administración de Sistemas Informáticos y Redes
2013-Actualidad Grado de Ingeniería Informática
Actualmente Cursando Tercer año del Grado de Ingeniería Informática

EXPERIENCIA LABORAL
2011 (julio-octubre) JCCE-Camagüey 2
Colaborador auxiliar de administrador de red

2013 Prácticas en ARINFO
Técnico de Adm. de Sistemas Informáticos y Redes.

IDIOMAS
CASTELLANO Lengua Materna
INGLÉS Nivel Medio oral y escrito.
2º Curso escuela de idiomas.

RESUMEN

Juego para tres jugadores que colocarán como máximo 2N reinas, y un jugador bloqueador que colocará N bloques en un tablero de tamaño 4N, ganará el jugador que más reinas coloque y en caso de empate ganan las negras o ganará el bloqueador si consigue que sus oponentes no sean capaces de colocar 4N/3 reinas.

INTRODUCCIÓN

Se ha desarrollado usando Jason Agent Speak. La estrategia a seguir por los jugadores ha sido colocar reinas en las posiciones libres que más casillas amenacen, de esta forma el espacio disponible para colocar es menor y la partida termina más rápidamente, el jugador con las reinas negras seguirá la misma estrategia ya que en cualquier caso depende de las malas elecciones de las blancas y/o del bloqueador para ganar, el bloqueador sigue la misma estrategia, colocar un bloque en la casilla libre que amenace más posiciones, intentando quitar la mejor jugada a los jugadores.

CÓDIGO COMPLETO DE LA PRÁCTICA

```
 // Agent r3 in project mars.mas2j

/* Initial beliefs and rules */

/* Initial goals */

!start.

/* Plans */
+!start : playAs(0) <-
  .wait(500);
  !amenazadas;
  !play
  .

+!start : playAs(1).


+!start : not playAs(_) <-
  .wait(500);
  !amenazadas;
  !putBlock.

+size(N)<-
  +blockNum(N/4);
  !crearTablero(N).

/* ----- Crea un tablero de casillas libres con el numero de casillas que amenaza cada una ----- */
+!crearTablero(N) <-
        for(.range(X,0,N-1)){
                for(.range(Y,0,N-1)){
                        +free(X,Y,N*N);
                }
        }.

+queen(X,Y) [source(percept)]  <-
        .print("Actualizando base de conocimientos");
        !ocupar(X,Y);
 // .findall(pos(PosX,PosY),free(PosX,PosY,_),Lista);
 // .print(Lista);
  .wait(1000);
        !amenazadas;
  .wait(1000);
  .
```

```
+block(X,Y) <-
  -free(X,Y,_);
          .print("Actualizando base de conocimientos");
  ?size(Size);
  for(.range(V,0,Size-1)){
    for(.range(W,0,Size-1)){
      if(not free(V,W,_) & not block(V,W) & not hole(V,W)){
        !check(V,W,Check);
        if(Check\==true & not hole(V,W)){
          +free(V,W,0);
          .print("Pos liberada",V,",",W);
        }
      }
    }
  }
  !amenazadas;
  .wait(1000);
  .findall(pos(PosX,PosY),free(PosX,PosY,_),Lista);
  .print(Lista)
 .

/* ----- Elimina casillas que no son libres ----- */
+!ocupar(X,Y) <-
  .print("Reina: ",X,",",Y);
          -free(X,Y,_);
  //Filas
  !filaDerecha(X+1,Y,ocupar);
  !filaIzquierda(X-1,Y,ocupar);

  //Columnas
  !columnaSuperior(X,Y-1,ocupar);
  !columnaInferior(X,Y+1,ocupar);

  //Diagonales
  !diagonalSI(X-1,Y-1,ocupar);
  !diagonalSD(X+1,Y-1,ocupar);
  !diagonalII(X-1,Y+1,ocupar);
  !diagonalID(X+1,Y+1,ocupar).
-!ocupar(X,Y).

/* ----- Comprueba que la casilla esté amenazada ----- */
+!check(X,Y,Check) <-
  if(queen(X,Y)){
    Check = true;
  }
```

```
  else {
    +check(false);
    //Filas
    !filaDerecha(X+1,Y,check);
    !filaIzquierda(X-1,Y,check);

    //Columnas
    !columnaSuperior(X,Y-1,check);
    !columnaInferior(X,Y+1,check);

    //Diagonales
    !diagonalSI(X-1,Y-1,check);
    !diagonalSD(X+1,Y-1,check);
    !diagonalII(X-1,Y+1,check);
    !diagonalID(X+1,Y+1,check);
    ?check(Check);
    .abolish(check(_));
  }
  .
-!check(X,Y)<-.print("ERROR CHECK").


//Filas
+!filaDerecha(X,Y,Z) : size(N) & not block(X,Y) & X<N <-
  if(Z == check & queen(X,Y) & check(false)){
    -+check(true);
  }
  if(Z == ocupar){
    -free(X,Y,_);
  }
  if(Z == contar & free(X,Y,_)){
    ?cont(AUX);
    -+cont(AUX+1);
   // .print(X,",",Y);
  }
  !filaDerecha(X+1,Y,Z).
+!filaDerecha(X,Y,Z).

+!filaIzquierda(X,Y,Z) : size(N) & not block(X,Y) & X>=0 <-
  if(Z == check & queen(X,Y) & check(false)){
    -+check(true);
  }
  if(Z == ocupar){
    -free(X,Y,_);
  }
  if(Z == contar & free(X,Y,_)){
```

```
    ?cont(AUX);
    -+cont(AUX+1);
   // .print(X,",",Y);
  }
  !filaIzquierda(X-1,Y,Z).
+!filaIzquierda(X,Y,Z).

//Columnas
+!columnaSuperior(X,Y,Z) : size(N) & not block(X,Y) & Y>=0 <-
  if(Z == check & queen(X,Y) & check(false)){
    -+check(true);
  }
  if(Z == ocupar){
    -free(X,Y,_);
  }
  if(Z == contar & free(X,Y,_)){
    ?cont(AUX);
    -+cont(AUX+1);
   // .print(X,",",Y,",",Z);
  }
  !columnaSuperior(X,Y-1,Z).
+!columnaSuperior(X,Y,Z).

+!columnaInferior(X,Y,Z) : size(N) & not block(X,Y) & Y<N <-
  if(Z == check & queen(X,Y) & check(false)){
    -+check(true);
  }
  if(Z == ocupar){
    -free(X,Y,_);
  }
  if(Z == contar & free(X,Y,_)){
    ?cont(AUX);
    -+cont(AUX+1);
   // .print(X,",",Y,",",Z);
  }
  !columnaInferior(X,Y+1,Z).
+!columnaInferior(X,Y,Z).

//Diagonales
+!diagonalSI(X,Y,Z) : size(N) & not block(X,Y) & Y>=0 & X>=0 <-
  if(Z == check & queen(X,Y) & check(false)){
    -+check(true);
  }
  if(Z == ocupar){
    -free(X,Y,_);
  }
```

```
  if(Z == contar & free(X,Y,_)){
    ?cont(AUX);
    -+cont(AUX+1);
   // .print(X,",",Y);
  }
  !diagonalSI(X-1,Y-1,Z).
+!diagonalSI(X,Y,Z).

+!diagonalSD(X,Y,Z) : size(N) & not block(X,Y) & Y>=0 & X<N <-
  if(Z == check & queen(X,Y) & check(false)){
    -+check(true);
  }
  if(Z == ocupar){
    -free(X,Y,_);
  }
  if(Z == contar & free(X,Y,_)){
    ?cont(AUX);
    -+cont(AUX+1);
   // .print(X,",",Y);
  }
  !diagonalSD(X+1,Y-1,Z).
+!diagonalSD(X,Y,Z).

+!diagonalII(X,Y,Z) : size(N) & not block(X,Y) & Y<N & X>=0 <-
  if(Z == check & queen(X,Y) & check(false)){
    -+check(true);
  }
  if(Z == ocupar){
    -free(X,Y,_);
  }
  if(Z == contar & free(X,Y,_)){
    ?cont(AUX);
    -+cont(AUX+1);
   // .print(X,",",Y);
  }
  !diagonalII(X-1,Y+1,Z).
+!diagonalII(X,Y,Z).

+!diagonalID(X,Y,Z) : size(N) & not block(X,Y) & Y<N & X<N <-
  if(Z == check & queen(X,Y) & check(false)){
    -+check(true);
  }
  if(Z == ocupar){
    -free(X,Y,_);
  }
  if(Z == contar & free(X,Y,_)){
```

```
    ?cont(AUX);
    -+cont(AUX+1);
   // .print(X,",",Y);
  }
  !diagonalID(X+1,Y+1,Z).
+!diagonalID(X,Y,Z).

/* ----- Actualiza el contador de casillas libres amenazadas ----- */
+!amenazadas <-
        ?size(N);
  +cont(0);
        for(free(X,Y,AM)){
    -+cont(0);
   // .print("ANALIZANDO ", X,",",Y);
    //Filas
    !filaDerecha(X+1,Y,contar);
    !filaIzquierda(X-1,Y,contar);

    //Columnas
    !columnaSuperior(X,Y-1,contar);
    !columnaInferior(X,Y+1,contar);

    //Diagonales
    !diagonalSI(X-1,Y-1,contar);
    !diagonalSD(X+1,Y-1,contar);
    !diagonalII(X-1,Y+1,contar);
    !diagonalID(X+1,Y+1,contar);

    ?cont(Amenazadas);
            -free(X,Y,AM);
    //La casilla X,Y se cuenta como amenazada tanto en filas como columnas como diagonales (-2)
            +free(X,Y,Amenazadas);
   // .print(X,",",Y,",",Amenazadas);
        }
  .abolish(cont(_)).
-!amenazadas<-.print("ERROR AMENAZADAS").

/* ----- Turnos ----- */

+player(N) : playAs(N) <- .wait(500); !play.

+player(N) : playAs(M) & not N==M /*& M\==blocker*/ <- .wait(300); .print("No es mi turno.").
```

```
/* ----- Jugar ----- */
+!play : blockNum(B) <-
  if (B > 0){
    .wait({+block(_,_)},1000,EventTime);
    -+blockNum(B-1);
    .wait(1000);
  } else {
    .wait(1000);
  }
  !select(Max);
  .print("Maximo: ", Max);
  !getPosition(Max, X,Y);
  queen(X,Y).
-!play <-.print("Juego Finalizado").

+!getPosition(pos(N,X,Y),X,Y).


/* ----- Seleccionar la Posición con mayor número de amenazadas ----- */
+!select(Max) <-
            .wait(700);
  //NumAmenazadas,X,Y
  .findall(pos(N,X,Y),free(X,Y,N),ListaPosiciones);
  .print("Posiciones posibles: ",ListaPosiciones);
  .max(ListaPosiciones,Max).
-!select(Max) <- Max = [].

/* ----- Colocar bloque ----- */
+!putBlock : blockNum(B) & B>0 <-
 // .print("BOQUESSSSSSSSSSSSSSSSS: ",B);
 // .random(R);
  .wait({+queen(_,_)});
  .wait(1000);
  !select(Max);
  .print("Maximo: ", Max);
  !getPosition(Max, X,Y);
  if(not queen(X,Y)){
    block(X,Y);
    -+blockNum(B-1);
  }
  !putBlock;
  .
-!putBlock <- .print("ERROR PUTBLOCK").
+!putBlock.
```

EXPLICACIÓN DEL CÓDIGO

Planes

```
+!start : playAs(0) <-
  .wait(500);
  !amenazadas;
  !play
  .
```

Comienza la partida para el turno del jugador 0
Calcula cuántas posiciones amenaza cada casilla libre y luego juega.

```
+!start : playAs(1).
```

Comienza la partida para el turno del jugador 1

```
+!start : not playAs(_) <-
  .wait(500);
  !amenazadas;
  !putBlock.
```

Comienza la partida para cuando no es turno de ninguno de los jugadores
Calcula cuántas posiciones amenaza cada casilla libre y luego juega como bloqueador.

```
+size(N)<-
  +blockNum(N/4);
  !crearTablero(N).
```

Una vez se conoce el tamaño del tablero se añade a la base de conocimientos el número de movimientos que podrá ejecutar el configurador y se crea un tablero con las dimensiones dadas.

```
+!crearTablero(N) <-
```

```
        for(.range(X,0,N-1)){
                for(.range(Y,0,N-1)){
                        +free(X,Y,N*N);
                }
        }.
```

Añade a la base de conocimientos todas las posiciones libres que tendrá el tablero inicialmente, es almacenado de la siguiente forma
free(X,Y,NumAmenazadas)
X indica la columna que ocupará en el tablero
Y indica la fila que ocupará en el tablero
NumAmenazadas indica el número de posiciones que amenazaría si se colocara una reina en X,Y. (Inicialmente es de N*N, ya que se actualiza con los valores reales antes del primer movimiento)

```
+queen(X,Y) [source(percept)]  <-
        .print("Actualizando base de conocimientos");
        !ocupar(X,Y);
 .wait(1000);
        !amenazadas;
 .wait(1000);
 .
```

En cada uno de los agentes, cuando se recibe una percepción de una reina se eliminan del tablero las casillas donde se ha colocado la reina y todas las que amenaza, luego se calcula cuántas posiciones amenaza cada casilla libre.

```
+block(X,Y) <-
 -free(X,Y,_);
        .print("Actualizando base de conocimientos");
 ?size(Size);
 for(.range(V,0,Size-1)){
  for(.range(W,0,Size-1)){
   if(not free(V,W,_) & not block(V,W) & not hole(V,W)){
    !check(V,W,Check);
    if(Check\==true & not hole(V,W)){
     +free(V,W,0);
```

```
        .print("Pos liberada",V,",",W);
      }
    }
  }
 }
 !amenazadas;
 .wait(1000);
 .findall(pos(PosX,PosY),free(PosX,PosY,_),Lista);
 .print(Lista)
.
```

En cada uno de los agentes, cuando se recibe una percepción de un bloque se elimina del tablero la casilla donde se ha colocado el bloque y "se libera" se añaden a la lista de casillas libres las posiciones que con la llegada del nuevo bloque dejan de estar amenazadas, luego se calcula cuántas posiciones amenaza cada casilla libre.

```
+!ocupar(X,Y) <-
 .print(X,",",Y);
        -free(X,Y,_);
 //Filas
 !filaDerecha(X+1,Y,ocupar);
 !filaIzquierda(X-1,Y,ocupar);

 //Columnas
 !columnaSuperior(X,Y-1,ocupar);
 !columnaInferior(X,Y+1,ocupar);

 //Diagonales
 !diagonalSI(X-1,Y-1,ocupar);
 !diagonalSD(X+1,Y-1,ocupar);
 !diagonalII(X-1,Y+1,ocupar);
 !diagonalID(X+1,Y+1,ocupar).
-!ocupar(X,Y).
```

Elimina del tablero las casillas amenazas (La posición donde se ha colocado la reina, y las posiciones que estén en la misma fila, columna y diagonales)

```
+!check(X,Y,Check) <-
  if(queen(X,Y)){
    Check = true;
  }
  else {
    +check(false);
    //Filas
    !filaDerecha(X+1,Y,check);
    !filaIzquierda(X-1,Y,check);

    //Columnas
    !columnaSuperior(X,Y-1,check);
    !columnaInferior(X,Y+1,check);

    //Diagonales
    !diagonalSI(X-1,Y-1,check);
    !diagonalSD(X+1,Y-1,check);
    !diagonalII(X-1,Y+1,check);
    !diagonalID(X+1,Y+1,check);
    ?check(Check);
    .abolish(check(_));
  }
  .
-!check(X,Y)<-.print("ERROR CHECK").
```

Comprueba si una casilla está amenazada (está amenazada si al realizar un recorrido en las filas, columnas y diagonales en todas las direcciones se encuentra alguna reina antes que un bloque o el final del tablero), en sea caso Check toma valor true; Check se almacena en la base de conocimientos y será true si cualquiera de las exploraciones lo determina así.

Los siguientes planes se explican juntos ya que cumplen una misma funcionalidad en diferentes "direcciones"

```
//Filas
+!filaDerecha(X,Y,Z) : size(N) & not block(X,Y) & X<N <-
  if(Z == check & queen(X,Y) & check(false)){
    -+check(true);
  }
  if(Z == ocupar){
```

```
    -free(X,Y,_);
  }
  if(Z == contar & free(X,Y,_)){
    ?cont(AUX);
    -+cont(AUX+1);
   // .print(X,",",Y);
  }
  !filaDerecha(X+1,Y,Z).
+!filaDerecha(X,Y,Z).

+!filaIzquierda(X,Y,Z) : size(N) & not block(X,Y) & X>=0 <-
  if(Z == check & queen(X,Y) & check(false)){
    -+check(true);
  }
  if(Z == ocupar){
    -free(X,Y,_);
  }
  if(Z == contar & free(X,Y,_)){
    ?cont(AUX);
    -+cont(AUX+1);
   // .print(X,",",Y);
  }
  !filaIzquierda(X-1,Y,Z).
+!filaIzquierda(X,Y,Z).

//Columnas
+!columnaSuperior(X,Y,Z) : size(N) & not block(X,Y) & Y>=0 <-
  if(Z == check & queen(X,Y) & check(false)){
    -+check(true);
  }
  if(Z == ocupar){
    -free(X,Y,_);
  }
  if(Z == contar & free(X,Y,_)){
    ?cont(AUX);
    -+cont(AUX+1);
   // .print(X,",",Y,",",Z);
  }
  !columnaSuperior(X,Y-1,Z).
```

```
+!columnaSuperior(X,Y,Z).

+!columnaInferior(X,Y,Z) : size(N) & not block(X,Y) & Y<N <-
  if(Z == check & queen(X,Y) & check(false)){
    -+check(true);
  }
  if(Z == ocupar){
    -free(X,Y,_);
  }
  if(Z == contar & free(X,Y,_)){
    ?cont(AUX);
    -+cont(AUX+1);
   // .print(X,",",Y,",",Z);
  }
  !columnaInferior(X,Y+1,Z).
+!columnaInferior(X,Y,Z).

//Diagonales
+!diagonalSI(X,Y,Z) : size(N) & not block(X,Y) & Y>=0 & X>=0 <-
  if(Z == check & queen(X,Y) & check(false)){
    -+check(true);
  }
  if(Z == ocupar){
    -free(X,Y,_);
  }
  if(Z == contar & free(X,Y,_)){
    ?cont(AUX);
    -+cont(AUX+1);
   // .print(X,",",Y);
  }
  !diagonalSI(X-1,Y-1,Z).
+!diagonalSI(X,Y,Z).

+!diagonalSD(X,Y,Z) : size(N) & not block(X,Y) & Y>=0 & X<N <-
  if(Z == check & queen(X,Y) & check(false)){
    -+check(true);
  }
  if(Z == ocupar){
    -free(X,Y,_);
```

```
  }
  if(Z == contar & free(X,Y,_)){
    ?cont(AUX);
    -+cont(AUX+1);
   // .print(X,",",Y);
  }
  !diagonalSD(X+1,Y-1,Z).
+!diagonalSD(X,Y,Z).

+!diagonalII(X,Y,Z) : size(N) & not block(X,Y) & Y<N & X>=0 <-
  if(Z == check & queen(X,Y) & check(false)){
    -+check(true);
  }
  if(Z == ocupar){
    -free(X,Y,_);
  }
  if(Z == contar & free(X,Y,_)){
    ?cont(AUX);
    -+cont(AUX+1);
   // .print(X,",",Y);
  }
  !diagonalII(X-1,Y+1,Z).
+!diagonalII(X,Y,Z).

+!diagonalID(X,Y,Z) : size(N) & not block(X,Y) & Y<N & X<N <-
  if(Z == check & queen(X,Y) & check(false)){
    -+check(true);
  }
  if(Z == ocupar){
    -free(X,Y,_);
  }
  if(Z == contar & free(X,Y,_)){
    ?cont(AUX);
    -+cont(AUX+1);
   // .print(X,",",Y);
  }
  !diagonalID(X+1,Y+1,Z).
+!diagonalID(X,Y,Z).
```

Los parámetros que admiten son X,Y,Z
X indica la columna de la casilla a analizar
Y indica la fila de la casilla a analizar
Z es utilizado para comprobar qué acción se realizará, puede ser ocupar o contar.
En cada uno se comprobará que los valores con los que se llama estén dentro de las dimensiones del tablero

Si Z es igual a "ocupar" se eliminará free(X,Y,_) de la base de conocimientos (la casilla ya no estará libre) y se ejecutará la misma meta recursivamente con la posición siguiente correspondiente a cada caso.
Por ejemplo filaDerecha(X,Y,ocupar) eliminará free X,Y de la base de conocimientos y ejecutará filaDerecha(X+1,Y,ocupar), y así sucesivamente hasta que no se cumpla que X esté dentro de las dimensiones del tablero.
De esta forma se avanza eliminado posiciones desde el punto donde se coloca la reina hasta el fin del tablero en cada uno de los sentidos.

Si Z es igual a "contar" se aumentará en 1 un contador en la base de conocimientos para cada casilla que sea libre por la que se pase. Actúa igual que el ejemplo mostrado anteriormete pero suma las casillas libres que hay en cada una de las direcciones y las va almacenando en una misma creencia, de forma que una vez terminado podamos consultar cuántas casillas quedarían amenazadas si colocáramos una reina en cierta posición.

Si Z es igual a "check" se comprueba si hay una reina en X,Y y que check sea falso, si todo esto se cumple se cambia el valor de Check en la base de conocimientos por true. Actúa igual que el ejemplo mostrado anteriormente pero comprueba si hay reinas antes de bloques o el fin del tablero en cada una de las direcciones.

```
+!amenazadas <-
        ?size(N);
  +cont(0);
        for(free(X,Y,AM)){
   -+cont(0);
   //Filas
   !filaDerecha(X+1,Y,contar);
   !filaIzquierda(X-1,Y,contar);

   //Columnas
   !columnaSuperior(X,Y-1,contar);
```

```
   !columnaInferior(X,Y+1,contar);

   //Diagonales
   !diagonalSI(X-1,Y-1,contar);
   !diagonalSD(X+1,Y-1,contar);
   !diagonalII(X-1,Y+1,contar);
   !diagonalID(X+1,Y+1,contar);

   ?cont(Amenazadas);
   -free(X,Y,AM);
   //La casilla X,Y se cuenta como amenazada tanto en filas como columnas como
diagonales (-2)
   +free(X,Y,Amenazadas);
  }
  .abolish(cont(_)).
-!amenazadas<-.print("ERROR AMENAZADAS").
```

Actualiza el tablero con el número de casillas que amenaza cada una de las posiciones libres. (free(X,Y,NumAmenazas))
Partiendo de cada una de las posiciones libres se añade una creencia a 0 (cont) y se recorre en cada uno de los sentidos contando y almacenando el número en dicha creencia, que luego se consultará para obtener el número de amenazadas en cada una de las posiciones, entonces se actualiza el valor y se elimina la creencia.

```
+player(N) : playAs(N) <- .wait(500); !play.

+player(N) : playAs(M) & not N==M <- .wait(300); .print("No es mi turno.").
```

Se comprueba que cada jugador juegue sólo durante su turno, si no es su turno envía un mensaje y espera.

```
/* ----- Jugar ----- */
+!play : blockNum(B) <-
  if (B > 0){
   .wait({+block(_,_)},1000,EventTime);
   -+blockNum(B-1);
   .wait(1000);
```

```
  } else {
    .wait(500);
  }
  !select(Max);
  .print("Maximo: ", Max);
  !getPosition(Max, X,Y);
  queen(X,Y).
-!play <-.print("Juego Finalizado").
```

Plan que se ejecuta durante el turno de cada jugador, comprueba si el valor de blockNum en la base de conocimientos es mayor que cero (si el configurador aún tiene movimientos disponibles), en ese caso espera a que se coloque un bloque o pase un segundo y se resta uno al número de movimientos del configurador en la base de conocimientos, en caso contrario simplemente espera medio segundo. Luego selecciona la casilla libre que amenace más posiciones y coloca una reina en ella.

```
+!getPosition(pos(N,X,Y),X,Y).
```

Obtiene los valores X e Y a partir de una entrada tipo pos(NumAmenazadas,X,Y)

```
+!select(Max) <-
        .wait(700);
  .findall(pos(N,X,Y),free(X,Y,N),ListaPosiciones);
  .print("Posiciones posibles: ",ListaPosiciones);
  .max(ListaPosiciones,Max).
-!select(Max) <- Max = [].
```

Selecciona el valor la casilla que amenace más posiciones libres:
Busca todas las posiciones libres (free(X,Y,NumAmenazadas)) y las almacena en una lista con la forma (pos(NumAmenazadas,X,Y)). Luego selecciona el valor máximo de dicha lista.

```
/* ----- Colocar bloque ----- */
+!putBlock : blockNum(B) & B>0 <-
 // .print("BOQUESSSSSSSSSSSSSSSSSS: ",B);
 // .random(R);
  .wait({+queen(_,_)});
  .wait(1000);
```

```
  !select(Max);
  .print("Maximo: ", Max);
  !getPosition(Max, X,Y);
  if(not queen(X,Y)){
    block(X,Y);
    -+blockNum(B-1);
  }
  !putBlock;
  .
-!putBlock <- .print("ERROR PUTBLOCK").
+!putBlock.
```

Plan mediante el cual **juega el configurador, se mantiene ejecutándose mientras el configurador disponga de movimientos (comprobando que blockNum en la base de conocimientos tenga un número mayor que 0). Espera a que se coloque una reina, se busca la posición libre donde si se colocara una reina amenazaría** más **posiciones, se coloca un bloque en ella y se reduce en uno en la base de conocimientos el número de movimientos** disponibles **para el** configurador**.**

CONCLUSIÓN

Los agentes funcionan correctamente y dentro de los parámetros establecidos para la práctica, no realizan ninguna jugada ilegal y la ejecución completa del código se realiza en un tiempo razonable.

REFERENCES

Manual oficial de la API Jason:
http://jason.sourceforge.net/api/overview-summary.html