Conocimiento y Razonamiento Automatizado

PECL1

Curso 2021-2022

Isabel Blanco Martínez

Laura Ramos Martínez

Patricia Cuesta Ruiz

ÍNDICE

REPARTO DE TAREAS

No ha habido un reparto claro de las tareas ya que durante la realización de toda la práctica en la gran mayoría de ocasiones nos hemos estado reuniendo las 3 integrantes del grupo para realizarla.

OBJETIVOS CUMPLIDOS

1. La solución propuesta debe basarse en el uso de listas.

La solución propuesta se basa en el uso de lista de filas. Sin embargo, para la consiguiente realización de la práctica, utilizaremos el método traspuesta [1] para trabajar con listas de columnas también.

Como se puede ver en este tablero de ejemplo que hemos utilizado para realizar pruebas:

```
tablero_prueba([
                ['_', '_', '_', '_', '_', '_', '_'],
                ['_', '_', '_', '_', '_', '_', '_'],
                ['_', '_', '_', '_', '_', '_', '_'],
                ['_', '_', '_', '_', '_', '_', '_'],
                ['_', '_', '_', '_', '_', '_', '_'],
                ['_', '_', '_', '_', '_', '_', '_']
                ]).
```

2. El tablero de juego se representa como una lista de listas.

El tablero que definimos se representa como una lista de las filas del tablero.

3. Al principio, el jugador verá un tablero en el que en todos los lugares del mismo están vacíos. Esto exige implementar un predicado que escriba en pantalla un tablero 6x7 que se irá actualizando según el desarrollo del juego.

El jugador verá un tablero en el que todas las casillas están vacías, es decir, "_". Para ello utilizamos el predicado *imprimir_mesa*.

```
  1   2   3   4   5   6   7
- - - - - - - - - - - - - - -
| _ | _ | _ | _ | _ | _ | _ |
- - - - - - - - - - - - - - -
| _ | _ | _ | _ | _ | _ | _ |
- - - - - - - - - - - - - - -
| _ | _ | _ | _ | _ | _ | _ |
- - - - - - - - - - - - - - -
| _ | _ | _ | _ | _ | _ | _ |
- - - - - - - - - - - - - - -
| _ | _ | _ | _ | _ | _ | _ |
- - - - - - - - - - - - - - -
| _ | _ | _ | _ | _ | _ | _ |
- - - - - - - - - - - - - - -
```

```
% IMPRIMIR MESA

imprimir_mesa([]):- nl.
imprimir_mesa(T,Lista_columnas):- imprimir_lista_espacios_aux(Lista_columnas), nl,
                                  generador_lista_guiones(15, L1), imprimir_lista(L1), nl,
                                  imprimir_tablero(T).
```

El predicado *imprimir_mesa* hace uso de otros predicados que explicaremos a continuación para imprimir el tablero completo:

```
% IMPRIMIR LISTA

imprimir_lista([]).
imprimir_lista([X|Y]):- write(X), write(' '), imprimir_lista(Y).
```

El predicado *imprimir_lista*, encargado de imprimir el número de cada una de las columnas que tendrá el tablero para facilitar las cosas al jugador.

```
% IMPRIMIR LISTA CON BARRA

imprimir_lista_con_barra([]).
imprimir_lista_con_barra([X|Y]):- write(X), write(' | '),
                                  imprimir_lista_con_barra(Y).
```

El predicado *imprimir_lista_con_barra* se encarga de imprimir las filas y columnas del tablero con las respectivas barras para dividir las casillas.

```
% IMPRIMIR TABLERO

imprimir_tablero([]):- nl.
imprimir_tablero([X|L]):- write('| '),
             imprimir_lista_con_barra(X), nl,
                       generador_lista_guiones(15,L1), imprimir_lista(L1), nl,
                       imprimir_tablero(L).
```

El predicado *imprimir_lista_espacios* se encarga de imprimir espacios entre los elementos de una lista. Lo usaremos para imprimir el número de columnas encima del tablero.

```
imprimir_lista_espacios([]).
imprimir_lista_espacios([X|Y]):- write('  '), write(X), write(' '), imprimir_lista_espacios(Y).

imprimir_lista_espacios_aux(Lista_espacios):- reverse(X, Lista_espacios),
                                              imprimir_lista_espacios(X).
```

El predicado *imprimir_tablero* nos permite imprimir el tablero sin el número de columnas indicado.

```
generador_tablero(Y, X, Out):- % Y = filas, X = columnas, Out = tablero
                               length(Out, Y),
                               maplist(length_list(X), Out).
```

El *generador_tablero* se encarga de generar el tablero con el número de filas y columnas pasadas por parámetro.

```
% GENERADOR GUIONES

generador_lista_guiones(N, Out):- % N = entero, Out = lista de salidatb
                                  length(Out, N), % Genera una lista out, tal que su longitud es N
                                  maplist(=(-), Out).
```

El predicado *generador_lista_guiones* nos permite añadir la cantidad de guiones que queramos. En este caso, lo utilizamos para separar la lista de columnas del tablero, del propio tablero.

4. Hay que definir un predicado de aridad cero (jugar) que incorpore la preparación del juego y que, además, llame al predicado (jugando) que se encargue de gestionar el desarrollo del juego. Este predicado jugando, entre otras cosas, es el que se ocupará de preguntar al jugador por el número de columna en el que desea introducir la ficha correspondiente, así como de indicar quién juega en cada momento.

Hemos creado el predicado de aridad cero llamado *jugar*.

```
jugar:- write('Introduzca el modo de juego deseado: '), nl,
        write('     1. Jugador contra jugador'), nl,
        write('     2. Jugador contra maquina tonta'), nl,
        write('     3. Jugador contra maquina menos tonta'), nl,
        write('     4. Visualizar partida (tienes que ser publico enloquecido)'), nl,
        write('        Estrategia simple -> jugardor x'), nl,
        write('        Estrategia avanzada -> jugador o'), nl,
        read(Modo_juego),
        write('Introduzca el numero de filas con las que quiere jugar: '), nl,
        read(Filas),
        write('Introduzca el numero de columnas con las que quiere jugar: '), nl,
        read(Columnas),
        generador_tablero(Filas, Columnas, Tablero),
        imprimir_mesa(Tablero,Lista_columnas), nl,
        gen_lista_columnas(Columnas, Lista_columnas),

        (
            Modo_juego == 1 -> jugando_JcJ(Tablero, Lista_columnas, 'x')
        ;
            Modo_juego == 2 -> jugando_maquina_simple(Tablero, Lista_columnas, 'x')
        ;
            Modo_juego == 3 -> jugando_maquina_avanzada(Tablero, Lista_columnas, 'x')
        ;
            Modo_juego == 4 -> jugando_maquinaVSmaquina(Tablero, Lista_columnas, 'x')
        ).
```

Este predicado pregunta al usuario el modo de juego en el cual quiere empezar la partida, y hará la llamada al predicado *jugando* correspondiente. Estos tres predicados son:

- *jugando_JcJ*

```
jugando_JcJ(Tablero, Lista_columnas, Jugador):- write('Comienza el turno del jugador '), write(Jugador), nl,
                                                leer_columna(Columna, Lista_columnas, Tablero),
                                                extraer_columna(Columna, Tablero, Col),
                                                introducir_ficha(Col, Jugador, Col2),
                                                introducir_col(Col2, Tablero, Columna, Tablero2),

                                                imprimir_mesa(Tablero2, Lista_columnas),
                                                (
                                                    comprobar_victoria(Jugador, Tablero2) -> write('                    '), nl,
                                                                                            write('                    '), nl,
                                                                                            write('                    '), nl,
                                                                                            write('                    '), nl,
                                                                                            write('                    '), nl,
                                                                                            write('                    '), nl, nl,
                                                                                            write('                    '), nl,
                                                                                            write('                    '), nl,
                                                                                            write('                    '), nl,
                                                                                            write('                    '), nl,
                                                                                            write('                    '), nl,
                                                                                            write('                    '), nl,
                                                                                            write('                    '), nl, nl,
                                                                                            write('          Ha ganado el jugador '), write(Jugador), nl, nl,
                                                                                            write('          EL PUBLICO ENLOQUECE!!!          '), nl, nl,
                                                                                            write('/ °_ʖ° / °_ʖ° / °_ʖ° \ °_ʖ° \ °_ʖ° \ / °_ʖ° / °_ʖ° / °_ʖ° \ °_ʖ° \ °_ʖ° \ / °_ʖ° / °_ʖ° / °_ʖ° \ °_ʖ° \ °_ʖ° \'), nl,
                                                                                            write('/ °_ʖ° / °_ʖ° / °_ʖ° \ °_ʖ° \ °_ʖ° \ / °_ʖ° / °_ʖ° / °_ʖ° \ °_ʖ° \ °_ʖ° \ / °_ʖ° / °_ʖ° / °_ʖ° \ °_ʖ° \ °_ʖ° \'), nl,
                                                                                            write('/ °_ʖ° / °_ʖ° / °_ʖ° \ °_ʖ° \ °_ʖ° \ / °_ʖ° / °_ʖ° / °_ʖ° \ °_ʖ° \ °_ʖ° \ / °_ʖ° / °_ʖ° / °_ʖ° \ °_ʖ° \ °_ʖ° \'), nl,
                                                                                            write('/ °_ʖ° / °_ʖ° / °_ʖ° \ °_ʖ° \ °_ʖ° \ / °_ʖ° / °_ʖ° / °_ʖ° \ °_ʖ° \ °_ʖ° \ / °_ʖ° / °_ʖ° / °_ʖ° \ °_ʖ° \ °_ʖ° \'), nl, nl

                                                ;
                                                    (
                                                        Jugador == 'x' -> Jugador2 = 'o'
                                                    ;
                                                        Jugador == 'o' -> Jugador2 = 'x'
                                                    ),
                                                    jugando_JcJ(Tablero2, Lista_columnas, Jugador2)
                                                ).
```

- *jugando_maquina_simple*

```
jugando_maquina_simple(Tablero, Lista_columnas, Jugador):- write('Comienza el turno del jugador '), write(Jugador), nl,
                                                          (
                                                              Jugador == 'x' -> leer_columna(Columna, Lista_columnas, Tablero)

                                                              ;

                                                              Jugador == 'o' -> respuesta_aleatoria(Lista_columnas, Tablero, Columna)
                                                          ),

                                                          extraer_columna(Columna, Tablero, Col),
                                                          introducir_ficha(Col, Jugador, Col2),
                                                          introducir_col(Col2, Tablero, Columna, Tablero2),

                                                          imprimir_mesa(Tablero2, Lista_columnas),
                                                          (
                                                              comprobar_victoria(Jugador, Tablero2) -> write('                    '), nl,
                                                                                                      write('                    '), nl,
                                                                                                      write('                    '), nl,
                                                                                                      write('                    '), nl,
                                                                                                      write('                    '), nl,
                                                                                                      write('                    '), nl, nl,
                                                                                                      write('                    '), nl,
                                                                                                      write('                    '), nl,
                                                                                                      write('                    '), nl,
                                                                                                      write('                    '), nl,
                                                                                                      write('                    '), nl,
                                                                                                      write('                    '), nl,
                                                                                                      write('                    '), nl, nl,
                                                                                                      write('          Ha ganado el jugador '), write(Jugador), nl, nl,
                                                                                                      write('          EL PUBLICO ENLOQUECE!!!          '), nl, nl,
                                                                                                      write('/ °_ʖ° / °_ʖ° / °_ʖ° \ °_ʖ° \ °_ʖ° \ / °_ʖ° / °_ʖ° / °_ʖ° \ °_ʖ° \ °_ʖ° \ / °_ʖ° / °_ʖ° / °_ʖ° \ °_ʖ° \ °_ʖ° \'), nl,
                                                                                                      write('/ °_ʖ° / °_ʖ° / °_ʖ° \ °_ʖ° \ °_ʖ° \ / °_ʖ° / °_ʖ° / °_ʖ° \ °_ʖ° \ °_ʖ° \ / °_ʖ° / °_ʖ° / °_ʖ° \ °_ʖ° \ °_ʖ° \'), nl,
                                                                                                      write('/ °_ʖ° / °_ʖ° / °_ʖ° \ °_ʖ° \ °_ʖ° \ / °_ʖ° / °_ʖ° / °_ʖ° \ °_ʖ° \ °_ʖ° \ / °_ʖ° / °_ʖ° / °_ʖ° \ °_ʖ° \ °_ʖ° \'), nl,
                                                                                                      write('/ °_ʖ° / °_ʖ° / °_ʖ° \ °_ʖ° \ °_ʖ° \ / °_ʖ° / °_ʖ° / °_ʖ° \ °_ʖ° \ °_ʖ° \ / °_ʖ° / °_ʖ° / °_ʖ° \ °_ʖ° \ °_ʖ° \'), nl, nl

                                                          ;
                                                              (
                                                                  Jugador == 'x' -> Jugador2 = 'o'
                                                              ;
                                                                  Jugador == 'o' -> Jugador2 = 'x'
                                                              ),
                                                              jugando_maquina_simple(Tablero2, Lista_columnas, Jugador2)
                                                          ).
```

- *jugando_maquina_avanzada*

```
jugando_maquina_avanzada(Tablero, Lista_columnas, Jugador):-  write('Comienza el turno del jugador '), write(Jugador), nl,
                                                             (
                                                                 Jugador == 'x' -> leer_columna(Columna, Lista_columnas, Tablero)

                                                                 ;

                                                                 Jugador == 'o' -> mejor_posicion(Tablero, Lista_columnas, Columna)
                                                             ),

                                                             extraer_columna(Columna, Tablero, Col),
                                                             introducir_ficha(Col, Jugador, Col2),
                                                             introducir_col(Col2, Tablero, Columna, Tablero2),

                                                             imprimir_mesa(Tablero2, Lista_columnas),
                                                             (
                                                                 comprobar_victoria(Jugador, Tablero2) -> write('                    '), nl,
                                                                                                         write('                    '), nl,
                                                                                                         write('                    '), nl,
                                                                                                         write('                    '), nl,
                                                                                                         write('                    '), nl,
                                                                                                         write('                    '), nl, nl,
                                                                                                         write('                    '), nl,
                                                                                                         write('                    '), nl,
                                                                                                         write('                    '), nl,
                                                                                                         write('                    '), nl,
                                                                                                         write('                    '), nl,
                                                                                                         write('                    '), nl,
                                                                                                         write('                    '), nl, nl,
                                                                                                         write('          Ha ganado el jugador '), write(Jugador), nl, nl,
                                                                                                         write('          EL PUBLICO ENLOQUECE!!!          '), nl, nl,
                                                                                                         write('/ °_ʖ° / °_ʖ° / °_ʖ° \ °_ʖ° \ °_ʖ° \ / °_ʖ° / °_ʖ° / °_ʖ° \ °_ʖ° \ °_ʖ° \ / °_ʖ° / °_ʖ° / °_ʖ° \ °_ʖ° \ °_ʖ° \'), nl,
                                                                                                         write('/ °_ʖ° / °_ʖ° / °_ʖ° \ °_ʖ° \ °_ʖ° \ / °_ʖ° / °_ʖ° / °_ʖ° \ °_ʖ° \ °_ʖ° \ / °_ʖ° / °_ʖ° / °_ʖ° \ °_ʖ° \ °_ʖ° \'), nl,
                                                                                                         write('/ °_ʖ° / °_ʖ° / °_ʖ° \ °_ʖ° \ °_ʖ° \ / °_ʖ° / °_ʖ° / °_ʖ° \ °_ʖ° \ °_ʖ° \ / °_ʖ° / °_ʖ° / °_ʖ° \ °_ʖ° \ °_ʖ° \'), nl,
                                                                                                         write('/ °_ʖ° / °_ʖ° / °_ʖ° \ °_ʖ° \ °_ʖ° \ / °_ʖ° / °_ʖ° / °_ʖ° \ °_ʖ° \ °_ʖ° \ / °_ʖ° / °_ʖ° / °_ʖ° \ °_ʖ° \ °_ʖ° \'), nl, nl

                                                             ;
                                                                 (
                                                                     Jugador == 'x' -> Jugador2 = 'o'
                                                                 ;
                                                                     Jugador == 'o' -> Jugador2 = 'x'
                                                                 ),
                                                                 jugando_maquina_avanzada(Tablero2, Lista_columnas, Jugador2)
                                                             ).
```

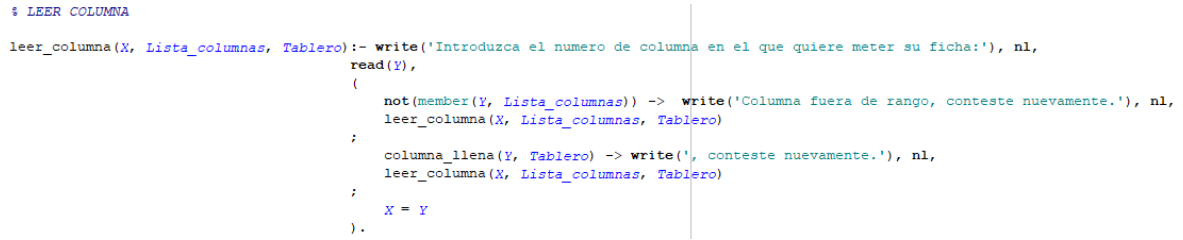
- *jugando_maquinaVSmaquina*

Cada uno de ellos gestiona el cambio de jugador y el dónde se insertan las fichas dependiendo de si se juega contra una persona o contra una de las máquinas creadas.

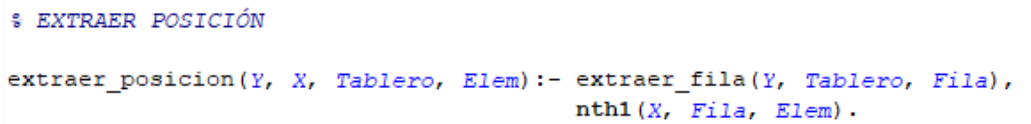
5. En cada turno de juego, se pueden dar dos casos:

    a. La columna tiene espacio libre: el jugador puede insertar su ficha que se situará al fondo de la columna.
    b. La columna está llena: el jugador debe elegir otra columna con espacio libre para insertar su ficha.

En cada turno, si la columna tiene espacio libre, la ficha se insertará en la columna que el usuario introduzca por teclado. Si es un número fuera del rango de las columnas disponibles pedirá al usuario introducir la columna en la que desea introducir la ficha, lo mismo sucede si la columna está llena, se volverá a pedir al usuario introducirlo por teclado.



El predicado *leer_columna* solicita una columna al jugador y comprueba que esta forme parte de la lista de columnas que forman el tablero, es decir, en este caso {1, 2, 3, 4, 5, 6, 7}. Si se sale de rango, vuelve solicitar una columna. Si la columna está llena, se vuelve a solicitar otra columna.



El predicado *extraer_posicion* nos permite obtener un elemento dadas una posición X e Y, el tablero, y el elemento que queremos retornar.

```
% EXTRAER FILA

extraer_fila(N, Tablero, Fila):- nth1(N, Tablero, Fila).
```

El predicado *extraer_fila* nos devuelve una lista con la fila deseada, pasándole el número de la fila requerida (N), el tablero y la Fila en cuestión.

```
% COMPROBAR SI UNA COLUMNA ESTÁ LLENA

columna_llena(X, Tablero):- extraer_posicion(1, X, Tablero, Elem),
                            (Elem \= '_') -> write('Columna llena').
```

El predicado *columna_llena* comprueba si la columna que se pasa por parámetro (X) está o no llena y comprobando si ese elemento es distinto de un hueco vacío (_).

6. Una vez insertada la ficha es necesario comprobar si se cumple alguna de las condiciones siguientes:

    a. La ficha insertada forma un grupo de 4 fichas en horizontal.
    b. La ficha insertada forma un grupo de 4 fichas en vertical.
    c. La ficha insertada forma un grupo de 4 fichas en cualquiera de las dos diagonales.

En el caso de nuestra práctica, comprobamos si la ficha insertada forma un grupo de 4 fichas en horizontal o en vertical.

Dada la complejidad de comprobar las diagonales no hemos sido capaces de realizarlo por nuestra cuenta, por lo que hemos encontrado un repositorio de GitHub [2] donde hace esa comprobación y lo hemos implementado en nuestra práctica.

```
% COMPROBAR VICTORIA: comprobar si algún jugador ha ganado la partida

comprobar_victoria(Jugador, Tablero):- comprobar_diagonal1(Jugador, Tablero);
                           comprobar_diagonal2(Jugador, Tablero);
                           comprobar_victoria_aux(Tablero); % comprueba filas
                           traspuesta(Tablero, TableroTras),
                           comprobar_victoria_aux(TableroTras). % comprueba Columnas

comprobar_victoria_aux([]):- false.
comprobar_victoria_aux([Fila|Resto]):- comprobar_fila(Fila, '_', 1); comprobar_victoria_aux(Resto).
```

El predicado *comprobar_victoria* se encarga de comprobar si *comprobar_diagonal1, comprobar_diagonal2*, *comprobar_victoria_aux* o *comprobar_victoria_aux* (con la traspuesta comprueba las columnas) devuelven True. En cuanto una de las condiciones anteriores se cumpla, el jugador ha ganado.

```
comprobar_fila([], _, _):- false.
comprobar_fila([Actual|Resto_fila], Anterior, Contador):- % imprimir_lista([Actual|Resto_fila]), write(Contador), nl,
                                                          (
                                                              Actual == Anterior, Actual \= '_' -> Contador2 is Contador+1, % si acutal es igual al anterior y dist
                                                              (
                                                                  Contador2 == 4 -> true % si el contador llega a 4, alguien gana
                                                              ;
                                                                  Contador2 =< 4 -> comprobar_fila(Resto_fila, Actual, Contador2) % si no es 4, seguimos buscando
                                                              )
                                                          ;
                                                              (Actual \= Anterior ; Actual == '_') -> Contador2 is 1, comprobar_fila(Resto_fila, Actual, Contador2)

                                                          ).
```

El predicado *comprobar_fila*, mediante un contador, comprueba que la posición actual tenga una ficha igual a la anterior y distinta de _ (posición libre). Si es igual, el contador aumenta y una vez llega a 4 devuelve true, ya que en esa fila encuentra 4 en raya. En caso de ser menor que 4, vuelve a llamar a *comprobar_fila* y seguimos buscando. Si la posición actual no es una ficha igual a la anterior o es igual a _, se resetea el contador y se comprueba el resto de filas.

Este predicado está hecho para comprobar 4 en raya para las filas. Sin embargo, si le pasamos la lista mediante la traspuesta, también podemos usarlo para comprobar las columnas.

```
% diagonal tipo \
comprobar_diagonal1(X, Tablero):- append(_, [C1,C2,C3,C4|_], Tablero),
                                  append(I1, [X|_], C1),
                                  append(I2, [X|_], C2),
                                  append(I3, [X|_], C3),
                                  append(I4, [X|_], C4),
                                  length(I1, M1), length(I2, M2), length(I3, M3), length(I4, M4),
                                  M2 is M1+1, M3 is M2+1, M4 is M3+1.

% diagonal tipo /
comprobar_diagonal2(X, Tablero):- append(_, [C1,C2,C3,C4|_], Tablero),
                                  append(I1, [X|_], C1),
                                  append(I2, [X|_], C2),
                                  append(I3, [X|_], C3),
                                  append(I4, [X|_], C4),
                                  length(I1, M1), length(I2, M2), length(I3, M3), length(I4, M4),
                                  M2 is M1-1, M3 is M2-1, M4 is M3-1.
```

Los predicados *comprobar_diagonal1* y *comprobar_diagonal2* comprueban si existen cuatro columnas en el tablero que contienen la ficha X. Comprueba las longitudes de las listas y va haciendo sumas o restas para ver si esas posiciones descienden (para el caso de las diagonales tipo \) o si ascienden (para el caso de las diagonales /).

7. En cualquiera de esos casos, el jugador que haya introducido la última ficha es el ganador de la partida.

Cuando se introduce una ficha que forma un grupo de 4 en horizontal, vertical o diagonal, ese jugador gana la partida.

En el predicado jugando, tras comprobar la victoria, en caso de que se cumpla alguna de las condiciones (4 en raya horizontal, vertical o diagonales) se imprimirá el mensaje de

8. Además del juego por turnos, se propone al alumno que desarrolle un modo de juego en el que el jugador se enfrente al ordenador de dos modos posibles:

    a. Estrategia simple: el ordenador elige una columna al azar en su turno.

Hemos realizado una estrategia simple que utiliza un **random** para decidir en qué columna meter la ficha.

```
respuesta_aleatoria(Lista_columnas, Tablero, Col_rand):- random_member(Col, Lista_columnas),
                                                         (
                                                             columna_llena(Col, Tablero) -> respuesta_aleatoria(Lista_columnas, Tablero, Col_rand)
                                                         ;
                                                             Col_rand = Col
                                                         ).
```

El predicado *respuesta_aleatoria* se encarga de seleccionar una columna aleatoria de la lista aleatoria de columnas del tablero. Comprueba si la columna está llena, y en caso de estarlo, vuelve a llamar a *respuesta_aleatoria* hasta seleccionar una columna que sea válida. En caso de ser válida, se introducirá la ficha en esa columna.

Este predicado se llamará en **jugando_maquina_simple**, como se puede ver en la imagen del objetivo 4. El jugador 'x', que es el usuario, seguirá introduciendo una columna introduciendola por teclado, y el jugador 'o' (la máquina) llamará al predicado *respuesta_aleatoria*.

Ya que este predicado se utiliza también en la estrategia avanzada, lo dejaremos en **tablero.pl**, en vez de incluirlo en el fichero **estrategia_simple.pl**.

    b. Estrategia avanzada: dotando de cierta inteligencia al proceso de toma de decisiones del ordenador.

Hemos optado por desarrollar una estrategia defensiva, que se basa en que cuando ve un grupo de 3 fichas enemigas intentar taparlo para evitar que gane. Esta estrategia se realiza en posiciones horizontales (de izquierda a derecha) y verticales, pero no lo hemos realizado para las diagonales.

Además, hemos implementado otra estrategia que reconoce cuando la máquina tiene 3 fichas seguidas y pondrá la ficha en la siguiente posición para hacer 4 en raya y ganar la partida.

Para ello, creamos los siguientes predicados:

- *mejor_posicion_horizontal*

```
% Devuelve la "mejor" posición en horizonal
mejor_posicion_horizontal([], Lista_columnas, _, Posicion_out):-length(Lista_columnas,N), Posicion_out is N+1.
mejor_posicion_horizontal([Fila_actual|Resto], Lista_columnas, Jugador, Posicion_out):- mejor_posicion_horizontal_aux(Fila_actual, Jugador, '_', 0, 0, Columna),
                                                                                        Columna2 is Columna+1,% como se meterá en la siguiente columna, sumamos 1
                                                                                        (
                                                                                            member(Columna2, Lista_columnas) -> Posicion_out = Columna2 %si la columna es va
                                                                                        ;
                                                                                            mejor_posicion_horizontal(Resto, Lista_columnas, Jugador, Posicion_out)
                                                                                        ).
```

Hemos dividido este método en dos, *mejor_posicion_horizontal* y *mejor_posicion_horizontal_aux*, el primero llamará recursivamente al segundo con todas las filas que comprenden el tablero, y cuando *mejor_posicion_horizonta_aux* encuentre una fila en la que haya tres o más fichas iguales seguidas devolverá esa posición sumado uno, ya que la columna en la que hay que introducir la ficha para evitar el cuatro en raya es la adyacente, con esto conseguimos que la máquina pueda atacar o defender en horizontal, atacará buscando fichas de su mismo tipo para completar los 4 en raya, y buscando fichas del enemigo defenderá evitando que consiga cuatro en raya.

```
% Basado punto a punto en el metodo de comprobar fila de tablero.pl
mejor_posicion_horizontal_aux([], _, _, _, Index_out, Index):- Index_out = Index. % asigna la salida al indice de la columna en la que acaba la sucesion de fichas enemigas
mejor_posicion_horizontal_aux([Actual|Resto_fila], Jugador, Anterior, Contador, Index, Index_out):- Index2 is Index+1,
                                                                                                    (
                                                                                                        Actual == 'x', Actual == Anterior, Actual \= '_' -> Contador2 is Contador+1, % actualizamos contador
                                                                                                        (
                                                                                                            Contador2 == 3 -> mejor_posicion_horizontal_aux([], _, _, _, Index2,Index_out) % llamada al caso base para
                                                                                                        ;
                                                                                                            Contador2 =< 3 ->  mejor_posicion_horizontal_aux(Resto_fila, Jugador, Actual, Contador2, Index2, Index_out)
                                                                                                        )
                                                                                                    ;
                                                                                                        (Actual \= Anterior ; Actual == '_') -> Contador2 is 1, mejor_posicion_horizontal_aux(Resto_fila, Jugador,
                                                                                                        Actual, Contador2,Index2, Index_out)

                                                                                                    ).
```

Mejor_posicion_horizonta_aux funciona usando un contador, que se incrementa cuando detecta fichas iguales seguidas y se reinicia cuando hay un cambio de ficha, si este contador llega a 3 en algún momento, retorna la posición de la lista en la que ha encontrado esa cadena de 3 fichas de un tipo concreto seguidas.

- *mejor_posicion_vertical*

```
mejor_posicion_vertical([],Lista_columnas,_,_,Contador_out):- length(Lista_columnas,N), Contador_out is N+1.
mejor_posicion_vertical([Columna_actual|Resto], Lista_columnas, Jugador, Contador, Contador_out):-(
                                                                                                    mejor_posicion_vertical_aux(Columna_actual, Jugador, '_', 0) -> Contador_out = Contador
                                                                                                ;
                                                                                                    Contador2 is Contador+1,
                                                                                                    mejor_posicion_vertical(Resto, Lista_columnas, Jugador, Contador2, Contador_out)
                                                                                                ).
```

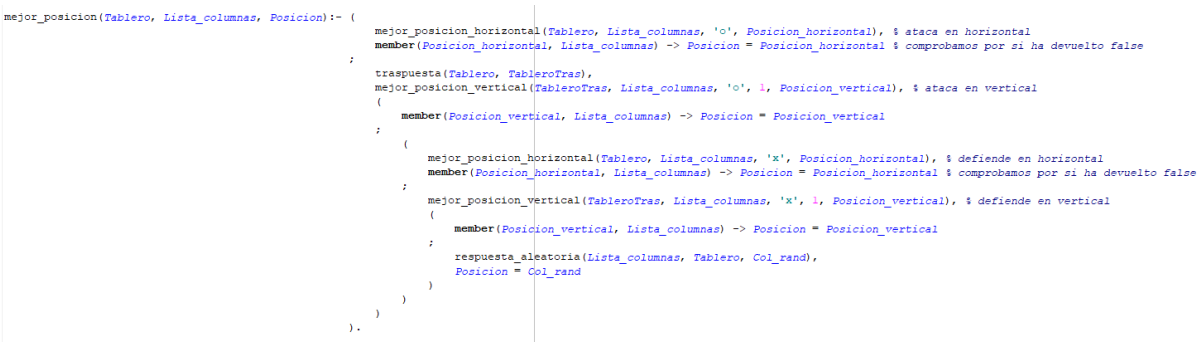
Este método funciona de manera similar a *mejor_posicion_horizontal*, salvo que en vez de guardar el contador de la función auxiliar (que recorre cada columna individualmente) y en vez lo guarda la función principal, sin sumarle uno, ya que para tapar una columna hemos de introducir ficha en esa misma columna.

```
% Basado punto a punto en el metodo de comprobar fila de tablero.pl
mejor_posicion_vertical_aux([], _, _, _):- false. %si la columna no tiene ninguna sucesion de 3 elementos enemigos, devuelve false
mejor_posicion_vertical_aux([Actual|Resto_fila], Jugador, Anterior, Contador):- (
                                                                                    Actual == 'x', Actual == Anterior, Actual \= '_' -> Contador2 is Contador+1,
                                                                                (
                                                                                    Contador2 == 3 -> true % si si hay tres elementos enemigos seguidos, los defiende
                                                                                ;
                                                                                    Contador2 =< 3 ->  mejor_posicion_vertical_aux(Resto_fila, Jugador, Actual, Contador2) %en caso contrario sigue buscando
                                                                                )
                                                                            ;
                                                                                (Actual \= Anterior ; Actual == '_') -> Contador2 is 1, mejor_posicion_vertical_aux(Resto_fila, Jugador, Actual, Contador2)

                                                                            ).
```

Mejor_posicion_vertical_aux tiene un funcionamiento casi identido a su contraparte horizontal, solo que en vez de retornar un entero si no encuentra una sucesión de 3 o más elementos iguales, retorna false, lo cual hace que la función superior *mejor_posicion_vertical* compruebe la siguiente columna.

- *mejor_posicion*



Mejor_posicion es el predicado que se encarga de coordinar las funciones anteriormente mencionadas, tiene una estrategia muy simple: primero intenta atacar, y si no hay una vía de ataque que detecte como viable, defiende, en caso de que tampoco haya algo de lo que defenderse, realizará un movimiento aleatorio. El metodo de control que usamos para saber si el movimiento que han devuelto las funciones individuales es valido es el predicado *member/2*, si la columna que han retornado no está dentro de la lista de columnas validas, retornará *false*, impidiendo que se siga ejecutando esa rama.

Siempre sigue el mismo orden para determinar su siguiente movimiento:

1. Ataca en horizontal.
2. Ataca en vertical.
3. Defiende en horizontal.
4. Defiende en vertical.
5. Realiza un movimiento aleatorio.

El predicado *mejor_posicion* se llamará en *jugando_maquina_avanzada*, como se puede ver en la imagen del objetivo 4. El jugador ‘x’, que es el usuario, seguirá introduciendo una columna introduciendo por teclado, y el jugador ‘o’ (la máquina) llamará al predicado *mejor_posicion*.

9. Estas estrategias se implementarán en ficheros separados (a consultar según proceda).

Todo el desarrollo de predicados se lleva a cabo en **tablero.pl**.

El desarrollo del modo de juego normal de **Jugador contra Jugador** se ha realizado en el fichero **jVSj.pl**.

La **estrategia simple** ha sido desarrollada en el fichero **estrategia_simple.pl**.

La **estrategia avanzada** se ha llevado a cabo en **estrategia_avanzada.pl**.

Además, la partida de máquina contra máquina se lleva a cabo en **maquinaVSmaquina.pl**.

TAREAS NO IMPLEMENTADAS

- La partida nunca comprueba el caso de un **posible empate**, ya que si todas las columnas están llenas, ninguno de los jugadores ganará ni podrá insertar más fichas, pero no finaliza la partida con un claro mensaje de 'empate'.
- Como hemos mencionado anteriormente, el código de la **comprobación de diagonales** está realizado pero no lo hemos programado nosotras.
- La estrategia avanzada de la máquina solo comprueba líneas horizontales de izquierda a derecha y no al revés, lo cual hace que solo cierre por el extremo derecho, así mismo, tras defender una serie de tres fichas enemigas, seguirá intentando defenderla insertando más fichas en esa misma columna.

MEJORAS REALIZADAS

1. Incluir elementos tipo ASCII art cuando gana uno de los jugadores. Debemos añadir que dicho ASCII art se puede ver ejecutandolo desde la terminal de Visual Studio Code, ya que SWI-Prolog-Editor no reconoce bien los caracteres y se muestra mal.

2. Hemos realizado la implementación de aumentar las dimensiones del tablero a medias, ya que el programa pregunta al usuario las dimensiones del tablero, pero no hemos realizado la implementación de que el número de fichas vaya aumentando consecuentemente.

```
Introduzca el numero de filas con las que quiere jugar:
|: 6.
Introduzca el numero de columnas con las que quiere jugar:
|: 7.
```

3. Hemos creado un modo de juego nuevo que enfrenta una máquina que utiliza la estrategia_simple y la estrategia_avanzada. Dicho modo de juego está implementado en el archivo **maquinaVsmaquina.pl**.

BIBLIOGRAFÍA

https://stackoverflow.com/questions/4280986/how-to-transpose-a-matrix-in-prolog

https://github.com/rvinas/connect-4-prolog/blob/master/connect4.pl

https://www.swi-prolog.org/pldoc/man?predicate=random_member/2