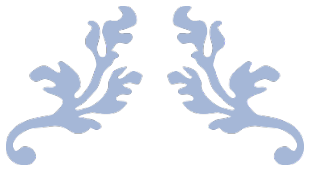


PL1
CONECTA4

CONOCIMIENTO Y RAZONAMIENTO AUTOMATIZADO

17 DE MARZO DE 2022

GRUPO DE LABORATORIO A3
Eduardo Ruiz Sabajanes, David Martínez Gutiérrez, Jesús Palomino Abreu

Contenido

REPARTO DE TAREAS

Para poder entender mejor el reparto de tareas, visualicemos el siguiente calendario:

	Sem 1	Sem 2	Sem 3	Sem 4	Sem 5
Jesús Palomino	1	4	7	12	15
Eduardo Ruiz	2	3	9, 10	11	14, 15
David Martínez	15	5, 6	8	13	15

En este calendario se puede apreciar por semanas, los distintos objetivos que se iban completando cada semana (cada número se corresponde con una de las actividades listadas más abajo), asociados al miembro del grupo cuya participación tuvo mayor importancia (ya sea porque lo completo totalmente por su cuenta, o porque llevó a cabo la tarea que supusiera una mayor dificultad para el objetivo en cuestión ya sea proponer el algoritmo a implementar, codificarlo, solucionar errores, etc.). Cabe destacar que no todas estas tareas suponían implementar alguno de los objetivos a alcanzar en la práctica, sino que, algunas de ellas representan la introducción de mejoras.

Con respecto a los distintos problemas y errores que nos hemos ido encontrando a lo largo del desarrollo de la práctica, estos problemas y errores, sin importar por quien fueran identificados, han sido solucionados por la misma persona que realizo la implementación problemática, pues entendemos que tiene la responsabilidad sobre dichos errores y es quien mejor entiende dicha implementación.

Por lo tanto, las actividades llevadas a cabo han sido:

1. **Creación del tablero**: esta tarea ha consistido en pensar en la estructura de datos que debemos usar para crear el tablero y, posteriormente, crear un predicado que devuelva ese tablero.
2. **Mostrar el tablero**: esta tarea ha consistido en crear los predicados necesarios para mostrar el tablero por pantalla.
3. **Cambios para adaptar el tablero a N filas y M columnas**: hubo un momento en el que dejamos de crear nuevos predicados, ya que nos dimos cuenta de que si queríamos añadir esta funcionalidad tendríamos que cambiar todos, así que procedimos a cambiar las dos anteriores tareas para que se adecúen a tableros de dimensiones diferentes.
4. **Insertar fichas**: todos los predicados necesarios para que el jugador elija una columna, se inserte la ficha en esa columna y se vuelva a mostrar el tablero con la ficha insertada.
5. **Turnos**: creación de los predicados utilizados para alternar los turnos entre ambos jugadores, comprobando después de cada uno si hay victoria o empate.
6. **Condiciones de victoria y empate**: creación de los predicados para comprobar si en el tablero tenemos un escenario de victoria, ya sea con los N elementos en fila, en columna o en diagonal; o un escenario de empate, es decir, un tablero completamente lleno.
7. **IA nivel fácil**: Se implementan de nuevo los turnos, pero esta vez para que el que juegue sea un bot. Este bot jugará siempre aleatoriamente.
8. **IA nivel difícil**: Se implementan los turnos para un bot que, en lugar de jugar aleatoriamente, seguirá una serie de condiciones: la IA elije primero una columna que le dé la victoria; si no existe la posibilidad, entonces elegirá una columna que evite la victoria del adversario; si tampoco es posible, jugará aleatoriamente.

9. **Adversario como parámetro**: Modificación del predicado *jugando()* para que se incluya el adversario como parámetro, de manera que sea más fácil plantear partidas, entre jugadores, jugador e IA y entre IAs.
10. **Estadísticas IA fácil vs IA difícil**: Creación de un modo de juego en el que jueguen las dos IAs un número N de partidas y se muestre por pantalla al final cuantas victorias ha conseguido cada una y cuantos empates ha habido.
11. **ASCII ART**: cambios en los caracteres del tablero para que sea más agradable visualmente. Además de dar color a las fichas que se introducen en el tablero.
12. **Menú principal**: tarea consistente en la creación de una especie de menú principal para que el programa te pregunte por el modo de juego (2 jugadores, IA fácil, IA difícil o IA vs IA), el número de filas, columnas y elementos a conectar, etc.
13. **Condición extra de elección IA difícil**: Aparte de las condiciones anteriores, quisimos añadirle un poco más de dificultad añadiéndole la siguiente condición: si no puede ganar ni evitar que el adversario gane, entonces jugará aleatoriamente pero solo en aquellas columnas que no hagan que en la siguiente jugada el adversario pueda ganar. Es decir, si al colocar en la columna 3, dejas un tablero en el que el adversario colocando en la columna 3 gana, esa columna la evitas.
14. **Modificación del formato del código**: se ha modificado el formato del código para que se vea más claro y se entienda mejor, además de separar las distintas funcionalidades del programa en distintos ficheros para no tener un fichero muy extenso. También se han intentado comprimir los predicados lo máximo posible para que quede un código más corto.
15. **EXTRAS**: Utilizamos EXTRAS para identificar aquellas tareas menos importantes o que hayan requerido menos esfuerzo como investigar cómo funcionan algunos predicados de Prolog, crear predicados auxiliares para la entrada de enteros o booleanos, etc.; o también para aquellas tareas consistentes en solucionar algunos bugs o errores que van ocurriendo al implementar nuevas funcionalidades.

GRADO DE CUMPLIMIENTO DE CADA UNO DE LOS REQUISITOS

SOLUCIÓN BASADA EN LISTAS

Nuestro tablero está formado por una lista de listas. Cada lista interna corresponde con una columna. Por lo tanto, tendremos una lista con tantas listas como columnas, las cuales a su vez tienen un tamaño del número de filas que queramos. A continuación, podemos ver el código de creación del tablero.

```
% board(Rows, Cols, X) -> Inicializa un tablero de Rows x Cols y lo
devuelve en X.
board(_, 0, []).
board(Rows, Cols, [C|XS]):-
    column(Rows, C),
    Cols2 is Cols-1,
    board(Rows, Cols2, XS).

% column(Rows, C) -> Inicializa una columna de Rows elementos y la
devuelve en C.
column(0, []).
column(Rows, [' '|CS]):-
    Rows2 is Rows-1,
    column(Rows2, CS).
```

Para inicializar el tablero llamamos a *board()*. Le pasamos el número de filas y columnas que queremos, y el resultado lo devolvemos en X. Será llamado una vez para cada columna, la cual se rellenará llamando a *column()*.

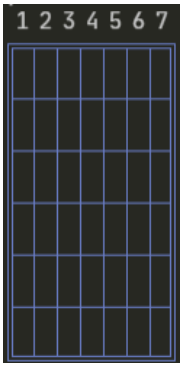
MOSTRAR TABLERO VACÍO AL COMIENZO

Como se puede ver más arriba, a la hora de rellenar cada columna introducimos el carácter espacio, por lo que, inicialmente, nuestro tablero vacío es un tablero lleno de espacios.

Para mostrar el tablero, tenemos definido en el fichero **mostrar_tablero.pl** la regla *show()*, cuyo resultado es el siguiente:

```
show(X, Rows, Cols):-
    writeHeader(Cols), nl,
    iShow(X, Rows, Cols, Rows).
```

Primero muestra los números de las columnas del tablero llamando a *writeHeader()*:




```
% writeHeader(Cols) -> Imprime por pantalla los números de las columnas
del tablero.
writeHeader(0).
writeHeader(Cols):-
```

```
    Cols2 is Cols-1,
    writeHeader(Cols2),
    write(' '), write(Cols).
```

Se llama Cols veces, e imprime un espacio y el número correspondiente a la columna.

Seguidamente se llama a *iShow()*, encargada de mostrar la tabla. Se itera tantas veces como filas tengamos, y en cada iteración llama primero a *dashLine()* y luego a *showLine()*.

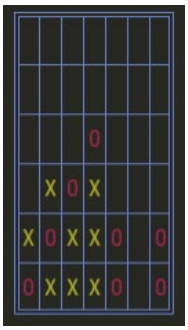
dashLine() muestra las filas que solo contienen bordes de la tabla. Contiene 3 subcasos en función de la columna en la que esté, y a su vez, otros 3 casos en función de la fila. De esta forma, como se puede ver en la imagen de arriba, los bordes de la tabla quedan dibujados con los símbolos Unicode correspondientes.

showLine() procede de forma similar, pero consultando la tabla, por lo que va a imprimir la ficha que corresponda al lugar que está imprimiendo.

Para imprimir los caracteres especiales, así como dar color, en lugar de *write()* usamos:

```
ansi_format([fg(blue)], '~w', '\u2502')
```

El resultado final de un tablero relleno sería el siguiente:



De esta forma somos capaces de mostrar tableros de cualquier tamaño de forma que el juego sea entendible.

IMPLEMENTACION DEL PREDICADO JUGAR

Nuestro predicado jugar despliega un menú para elegir el modo de juego, filas, columnas, elementos a conectar para ganar. También pregunta por qué tableros mostrar en los modos de juego en los que participa la IA. Una vez se tienen todos los datos, inicia el juego.

Para ello lo primero que haremos será consultar los ficheros donde definimos los predicados que vamos a utilizar:

```
jugar:-
    consult('entradas.pl'),
    consult('fin_juego.pl'),
    consult('introducir_ficha.pl'),
    consult('mostrar_tablero.pl'),
    gameMode(Player1, Player2, M),
    tableRows(R),
    tableColumns(C),
    elementsConnected(R, C, E),

    ...
```

El primer predicado al que llamaremos será *gameMode()*. Este nos devuelve cuales van a ser nuestros jugadores y el modo de juego seleccionado:

```
gameMode(Player1, Player2, M):-
    repeat,
    write_ln('Elige el modo de juego:'),
    write_ln('1. Jugar contra otro jugador'),
    write_ln('2. Jugar contra un bot (Fácil)'),
    write_ln('3. Jugar contra un bot (Difícil)'),
    write_ln('4. Enfrentar bots (Fácil Vs. Difícil)'),
    write_ln('5. Enfrentar bots (Estadísticas)'),
    read(M),
    integer(M),
    0 < M,
    M < 6,
    (
        (
            M = 1,
            Player1 = jugando,
            Player2 = jugando
        );
        (
            M = 2,
            Player1 = jugando,
            Player2 = jugandoIAFacil,
            consult('ia_facil.pl')
        );
        (
            M = 3,
            Player1 = jugando,
            Player2 = jugandoIADificil,
            consult('ia_dificil.pl')
        );
        (
            M > 3,
            Player1 = jugandoIAFacil,
            Player2 = jugandoIADificil,
            consult('ia_facil.pl'),
            consult('ia_dificil.pl')
        )
    ).
```

Comenzamos imprimiendo por pantalla nuestro menú. Leeremos la opción elegida por el usuario y tratamos de convertirla en un entero, para luego comprobar que está en el rango de opciones posibles. Por lo tanto, si la entrada no es válida, repetiremos el proceso hasta que lo sea. En la opción ambos jugadores son personas, por lo tanto, su turno se realizará siguiendo las reglas definidas en jugando. En la segunda, el jugador uno será una persona, y el 2 la IA fácil. Finalmente, el modo 3 es la IA fácil contra la difícil. Cuando vayamos a utilizar una IA nos aseguramos de consultar sus predicados para usarlos posteriormente.

Una vez tenemos el modo de juego y el tipo de jugadores, pasaremos a preguntarle al jugador por las reglas del juego. Comenzaremos con *tableRows()*:

```
tableRows(R):-
    repeat,
    write_ln('\u00bfCuantas filas va a tener el tablero?'),
    read(R),
    integer(R),
    0 < R.
```

Le preguntamos al usuario hasta que comprobamos que en efecto es un número de filas válido.

tableColumns() y *elementsConnected()* tienen una estructura similar. Por lo tanto, por ahora ya tenemos el tipo de jugadores de la partida (IA o Humano), las dimensiones del tablero y el número necesario de fichas necesario para ganar. Procedemos a analizar la segunda parte de jugar:

```
...

(
        (
            M = 4,
            showAllBoards(S),
            game(R, C, E, Player1, Player2, _, S, true)
        );
        (
            M = 5,
            simulationNumber(N),
            showFinalBoards(S),
            startSimulation(N, R, C, E, 0, 0, 0, S)
        );
        game(R, C, E, Player1, Player2, _, false, true)
    ).
```

Si el modo es el 4, mediante *showAllBoards()*, el usuario nos indicará si habrá que mostrar todos los movimientos de las partidas o solo el final. Esto se pasará mediante el parámetro S.

Para el modo de juego 5, *simulationNumber()* nos da el número de partidas que se jugarán entre las 2 IAs, *showFinalBoards()* si se tendrán que mostrar los tableros finales de cada una de las partidas o si, al contrario, solo se mostrarán las estadísticas de victorias de cada una de las IAs.

Finalmente, en el modo 5 se llama a startSimulation la cual luego analizaremos, en la que internamente se llama a *game(),* y en el resto de los modos se le llama directamente en *jugar()*.

```
game(Rows,Cols,Elems,Player1,Player2,Winner,ShowBoard,ShowFinalBoard):-
    board(Rows, Cols, X),

call(Player1,X,'X',Rows,Cols,Elems,'O',Player2,Winner,ShowBoard,ShowFinalBoard
).
```

Game() recibe los siguientes parámetros: *Rows*, *Cols* y *Elems* (características de las partidas), *Player1* y *Player2*, el tipo de jugador que se va a llamar, *ShowBoard* y *ShowFinalBoard* ya explicados anteriormente. *Winner* será un parámetro de retorno utilizado en *startSimulation().*

Primero se llama a *board()* para crear el tablero de la partida, y luego llamamos a *call()*. Este predicado propio de Prolog llama a un predicado (1º parámetro) con los parámetros definidos a partir del 2º. Esto lo hacemos por el siguiente motivo: Si *Player1* se ha elegido como jugador, tendrá el valor ***jugando***, por lo que *call()* llamará al predicado *jugando()*, que realizará las acciones correspondientes al turno de un jugador (pidiendo por entrada la posición de la ficha). Las otras opciones son que se *Player1* sea *jugandoIAFacil* (corresponde con la IA más sencilla) y *jugandoIADificil*, correspondiendo con la IA más compleja.

Vamos a analizar el predicado *jugando()*:

```
jugando(X, Player, Rows, Cols, N, Opponent, NextTurn, _, ShowBoard,
ShowFinalBoard):- % Juega
    show(X, Rows, Cols),
    write('Turno del jugador '),
    (
        (
```

```
            Player = 'X',
            ansi_format([fg(yellow)], '~w', 'X')
        );
        (
            Player = 'O',
            ansi_format([fg(red)], '~w', 'O')
        )
    ),
    nl,
    repeat,
    readColumn(C, Cols),
    insert(X, C, Player, X2),
    nl,

...
```

Comenzamos mostrando el tablero, y de quién es el turno, si de 'O' o de 'X'. Luego le pedimos al jugador que introduzca la columna en la que se va a insertar la ficha, hasta que se introduzca de forma correcta (*readColumn()*). La ficha se inserta llamando al predicado *insert()*:

```
insert([C|XS], 1, Elem, [C2|XS]):-
    insertColumn(C, Elem, C2).
insert([C|X], N, Elem, [C|X2]):-
    Ns is N-1,
    insert(X, Ns, Elem, X2).

insertColumn([' '], Elem, [Elem]).
insertColumn([' ',X|CS], Elem, [Elem,X|CS]):-
    not(X = ' ').
insertColumn([' ',' '|CS], Elem, [' '|C2]):-
    insertColumn([' '|CS], Elem, C2).
```

insert() recibe el tablero *X* en el que se va a insertar la ficha, *N* la columna en la que se va a insertar, *Elem* el carácter correspondiente con la ficha a introducir, y *X2* será el tablero con la ficha ya introducida.

Vamos a extraer la primera columna reduciendo consecuentemente *N*, hasta que *N* sea 1, es decir, que tengamos en *C* la columna en la que deseamos insertar el elemento. Una vez tenemos *C*, llamamos a *insertColumn()*. El tercer predicado es el caso en el que tenemos 2 espacios vacíos al principio de la columna. Vamos quitando estos espacios hasta que encontramos uno de los 2 casos base representados en los primeros 2 predicados. El primero es que ya solo queda un elemento vacío, que significa que la columna está vacía, en cuyo caso introducimos la ficha abajo del todo. El segundo caso es que ya haya fichas puestas, por lo tanto, colocaremos el elemento encima de esta. Tras colocar la ficha recomponemos tanto la columna como el tablero en *C2* y *X2* correspondientemente.

```
...
    (
        (
            win(Player, X2, N), % Gana
            show(X2, Rows, Cols),
            write('El jugador '),
            (
                (
                    Player = 'X',
                    ansi_format([fg(yellow)], '~w', 'X')
                );
                (
```

```
                    Player = 'O',
                    ansi_format([fg(red)], '~w', 'O')
                )
            ),
            write_ln(' ha ganado!')
        );
        (
            full(X2, Cols), % Empata
            show(X2, Rows, Cols),
            write_ln('Empate!')
        );
        call(NextTurn, X2, Opponent, Rows, Cols, N, Player, jugando,
_, ShowBoard, ShowFinalBoard) % Continua
    ).
```

Continuando con jugando, tras insertar una ficha comprobaremos si hemos ganado. Para ello llamamos al predicado *win()* el cual explicamos más adelante. Si se ha ganado, se mostrará por pantalla esta información.

Si lo anterior no se cumple, comprueba si el tablero está lleno mediante el predicado *full()*. En este caso significará que se ha empatado, y procederá a mostrarse esta información.

```
full([C|_], 1):-
    fullColumn(C).
full([C|XS], Cols):-
    fullColumn(C),
    Cols2 is Cols-1,
    full(XS, Cols2).

fullColumn([E|_]):-
    not(E = ' ').
```

Para comprobar si está lleno el tablero, lo desmontamos en columnas y comprobamos para cada una si contiene algún espacio. En caso de contenerlo, no está lleno y el predicado sería falso.

Finalmente, si no se cumplen los casos anteriores significa que el juego no ha terminado y debe seguir. Para ello llamamos a *call()*, pero esta vez intercambiado *Player1* y *Player2*, llamando al predicado almacenado en *NextTurn*.

De esta forma vamos alternando entre el predicado *Player1* y *Player2* hasta que termina la partida. Esto nos permite que, independientemente de si el juego será jugador vs IA o IA vs IA o cualquier combinación, podremos reutilizar estos predicados y solo implementar el predicado que defina cómo y dónde se inserta la ficha ese jugador. En el caso de un jugador, como hemos visto, se introducirá por consola, y en el caso de las IAs seguirán su propio criterio como ahora veremos.

COMPROBACIÓN DE INSERTAR

Como ya hemos visto, a la hora de introducir una ficha hacemos un *repeat*, que ejecutará *readColumn()* e *insert()* hasta que ambas sean ciertas, es decir, la columna donde se va a introducir la ficha existe y, en efecto, se haya insertado el elemento. Es el propio *insert()* el que si no se ha introducido ficha debido a que no había sitio en esa columna nos devuelve falso.

UNA VEZ INSERTADA LA FICHA SE COMPROBARÁ SI HA GANADO

Ya hemos mencionado el predicado *win()*. Este está definido en el fichero **fin_juego.pl**. En este fichero también se encuentra el ya explicado *full()*.

```
win(E, [C|XS], N):-
    winRow(E, [C|XS], _, N);    %-
    winCol(E, C, N, N);         %|
    winDiag1(E, [C|XS], _, N); %/
    winDiag2(E, [C|XS], _, N); %\
    win(E, XS, N).
```

win() recibe *E*, el símbolo correspondiente al jugador que se desea comprobar si ha ganado, el tablero *X* y *N*, el número de elementos necesarios para ganar.

Procederemos a comprobar si en las *N* primeras columnas se da uno de los siguientes casos: *winRow()*, hay una fila con *N* elementos seguidos, *winDiag1()*, hay una diagonal de *N* elementos, *winDiag2()*, similar al anterior pero la contraria. *WinCol()* comprueba una sola columna.

Volvemos a llamar a *win()* pero quitándole a *X* la primera columna y repetiremos el proceso hasta llegar a la última columna. Vamos a ver cómo funciona cada una de estas comprobaciones en profundidad.

```
winCol(_, _, 0, _).
winCol(E, [Elem|CS], N, NTot):-
    (
        E = Elem,
        N2 is N-1,
        winCol(E, CS, N2, NTot),
        !
    );
    winCol(E, CS, NTot, NTot).
```

winCol() recibe una lista y la va dividiendo hasta que está vacía o hasta que encuentra *N* elementos seguidos, en cuyo caso no comprueba más posibilidades (uso del operando !).

```
winRow(_, _, _, 0).
winRow(E, [C|XS], L, N):-
    append(E1, [E|_], C),
    length(E1, L),
    N2 is N-1,
    winRow(E, XS, L, N2).
```

winRow() busca en una columna la primera aparición del elemento *E*, y guarda la longitud de la lista de elementos que tenga encima. Comprobamos si las *N* columnas que estamos comprobando tengan un elemento *E* en esa posición y si se cumple devolvemos verdadero. Si no, se comprobará con elementos *E* que estuvieran en una profundidad mayor.

```
winDiag1(_, _, _, 0).
winDiag1(E, [C|XS], L, N):-
    append(E1, [E|_], C),
    length(E1, L),
    N2 is N-1,
    L2 is L-1,
    0 =< L2,
    winDiag1(E, XS, L2, N2).
```

winDiag1() y *winDiag2()* funcionan de forma similar a *winRow()*, pero teniendo en cuenta que la altura no tendrá que ser la misma, si no aumentar o disminuir 1 en cada columna.

SE IMPLEMETARÁN 2 IAs, UNA SIMPLE Y OTRA AVANZADA

La estrategia simple está implementada por el predicado *jugandoIAFacil()*. Imprimirá el tablero o no en función del parámetro *ShowBoard().* Elegirá un número aleatorio entre las columnas (sustituyendo a la entrada por consola de *jugando()*) y procederá de manera similar. Si la columna está llena se elegirá un número de nuevo hasta que haya una que sea válida. Para elegir el número aleatorio utilizamos el predicado *random_between()*.

En definitiva, es similar al predicado *jugando()*, pero recibiendo el parámetro *ShowFinalBoard()*, que decide si se mostrará el tablero si se ha llegado al final de la partida, y *ShowBoard()* que lo decide para cada turno.

```
    repeat,
    random_between(1, Cols, R),
    insert(X, R, Player, X2),
```

La estrategia avanzada está implementada en el predicado *jugandoIADificil()*. Similar a jugandoIAFacil(), pero en vez de elegir un número aleatorio directamente, se llama al predicado *playIAWin()*.

```
playAIWin(X, Cols, 0, N, X2, Player, Opponent) :-
    playAIAvoid(X, Cols, Cols, N, X2, Player, Opponent).
playAIWin(X, Cols, Column, N, X2, Player, Opponent):-
    (
        insert(X, Column, Player, X2),
        win(Player, X2, N)
    );
    (
        Column2 is Column - 1,
        playAIWin(X, Cols, Column2, N, X2, Player, Opponent)
    ).
```

Probamos metiendo una ficha en cada una de las columnas y ver comprobamos si se ha ganado haciendo ese movimiento. Si se gana, se introduce esa ficha. Si no se pasa a la siguiente columna hasta que nos quedemos sin, en cuyo caso llamamos *playAIAvoid()*, el cual intentará evitar que el adversario gane en el siguiente turno.

```
playAIAvoid(X, Cols, 0, N, X2, Player, Opponent):-
    playAIWontLose(X, Cols, Cols, N, X2, Player, Opponent, L),
    playAIRandom(X, Cols, X2, Player, L).
playAIAvoid(X, Cols, Column, N, X2, Player, Opponent):-
    (
        insert(X, Column, Opponent, X3),
        win(Opponent, X3, N),
        insert(X, Column, Player, X2)
    );
    (
        Column2 is Column - 1,
        playAIAvoid(X, Cols, Column2, N, X2, Player, Opponent)
    ).
```

Por lo tanto, comprueba columna a columna si ganase el oponente insertando en esa columna. Por lo tanto, coloca ficha donde ganaría el oponente para evitar que gane. Es por eso por lo que inserta el oponente, devuelve *X3* (tablero donde habría insertado el oponente) y comprobamos si ha ganado en ese tablero. Luego insertamos en esa misma posición en el tablero *X*, guardando en *X2* el tablero resultado de haber introducido la ficha. Si no se da este caso, se llamará a *playIAWontLose()*.

```
playAIWontLose(_, _, 0, _, _, _, _, []).
playAIWontLose(X, Cols, Column, N, X2, Player, Opponent, L):-
    Column2 is Column - 1,
    playAIWontLose(X, Cols, Column2, N, X2, Player, Opponent, LS),
    (
        (
            insert(X, Column, Player, X3),
            insert(X3, Column, Opponent, X4),
            not(win(Opponent, X4, N)),
            L = [Column|LS]
        );
        (
            insert(X, Column, Player, X3),
            nth1(Column, X3, C3),
            fullColumn(C3),
            L = [Column|LS]
        );
        L = LS
    ).
```

Aquí lo que hacemos es colocar una ficha en una columna, y comprobamos si el contrincante colocando en la misma columna que acabamos de colocar ficha (es la única posición que ha cambiado) va a ganar. Por lo tanto, se devuelve una lista con las posiciones en las que se puede colocar ficha sin que el contrincante gana. *PlayAIRandom()* se encargará de elegir una ficha de esta lista, por lo que finalmente si no se dan las condiciones mencionadas, se coloca ficha de forma aleatoria entre las posiciones que no hacen al otro ganar inmediatamente. Esto lo hacemos para que no se elija siempre la primera posición que cumple esta condición, y sea más variado.

```
playAIRandom(X, Cols, X2, Player, L):-
    length(L, Len),
    (
        (
            Len = 0,
            repeat,
            random_between(1, Cols, R),
            insert(X, R, Player, X2)
        );
        (
            repeat,
            random_member(R, L),
            insert(X, R, Player, X2)
        )
    ).
```

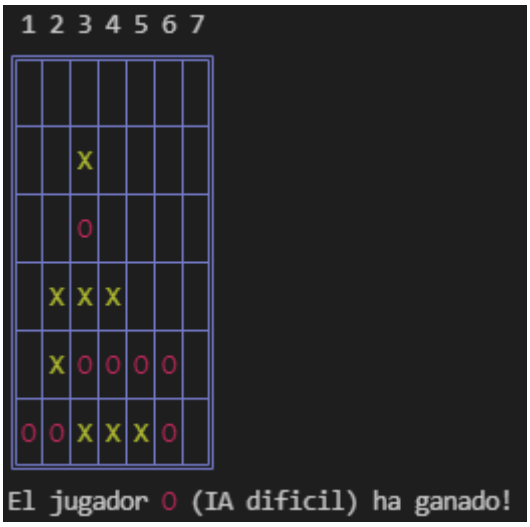
ESTAS ESTRATEGIAS SE IMPLEMENTARÁN EN FICHEROS SEPARADOS

Los predicados descritos respectivos a la estrategia simple están definidos en el archivo **ia_facil.pl**, y la estrategia más avanzada en **ia_dificil.pl**. Se importan como ya hemos mencionado anteriormente mediante *consult()*.

MEJORAS REALIZADAS

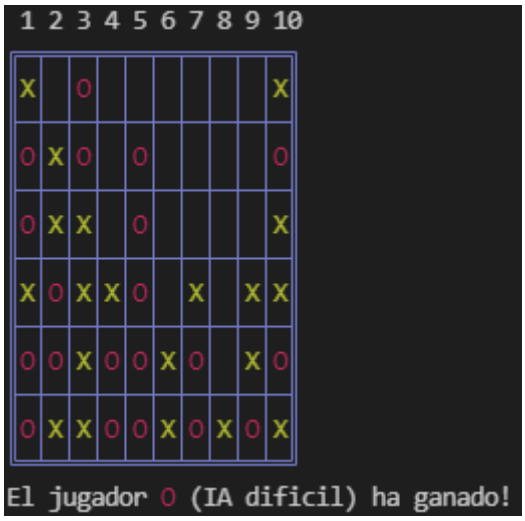
MEJORAR LA INTERFAZ CON ALGÚN ELEMENTO DE TIPO 'ASCII ART'

Como ya hemos visto anteriormente, decoramos el tablero utilizando caracteres Unicode, así como agregando colores (lo cual solo funciona en determinadas consolas) lo que mejora la legibilidad. E aquí un ejemplo:



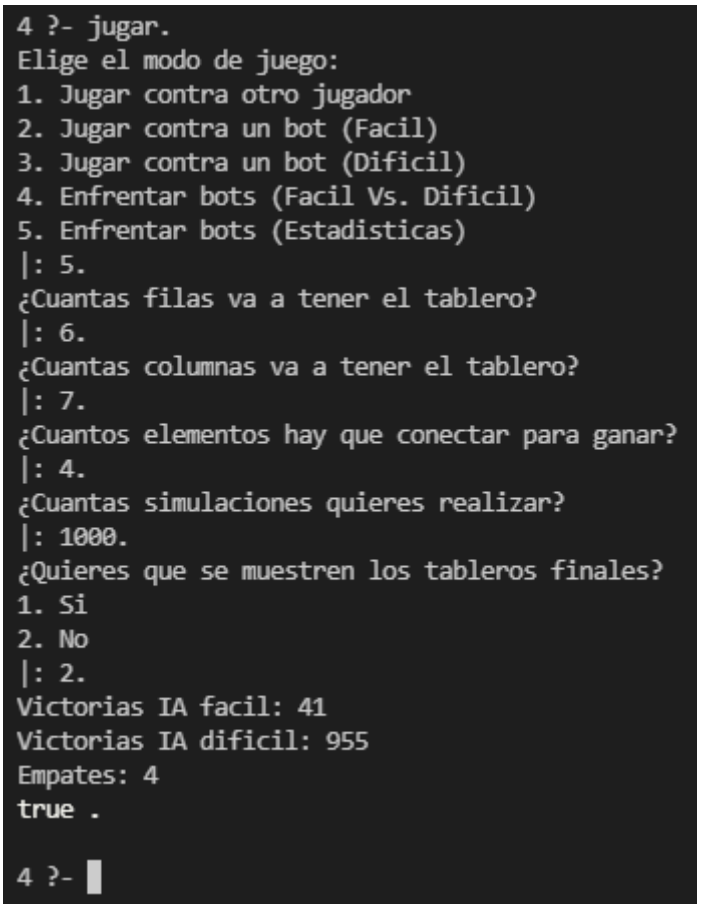
AUMENTAR LAS DIMENSIONES DEL TABLERO Y FICHAS NECESARIAS PARA GANAR

Como se ha visto a través de toda la memoria, está todo pensado para permitir cualquier número de filas y columnas, así como para poder decidir el número de fichas necesarias para ganar la partida. Aquí vemos un ejemplo de una partida entre IAs en un tablero de 6x10 con 5 fichas necesarias para ganar:



EVALUAR LA ESTRATEGIA SIMPLE CONTRA LA AVANZADA

Par evaluar las dos estrategias tenemos el modo de juego 5. Se lleva a cabo una simulación del número de partidas deseado, pudiendo configurar las características del tablero como se ve en el siguiente ejemplo de ejecución.



Como podemos ver, los resultados son los esperados. Un 95.5% de las partidas fueron victorias de la estrategia más avanzada, frente a tan solo un 4.1% de victorias de la IA fácil. Esto demuestra que, en efecto hemos cumplido nuestro objetivo y la avanzada es mejor. También se ve que un 0.4% de las partidas han acabado en empate.

FUENTES

Para la realización de la práctica se han consultado las siguientes fuentes:

- https://github.com/rvinas/connect-4-prolog
  - Se utilizo este repositorio para ver otro acercamiento al problema propuesto que nos diese algo de orientación sobre como empezar a realizar la práctica ya que no estábamos familiarizados con el lenguaje y en especial el uso de listas.
- https://unicode-table.com/es/
  - Se utilizó esta página para conocer los códigos Unicode de los caracteres con los que dibujamos el tablero de juego.
- https://www.swi-prolog.org/pldoc/doc_for?object=manual
  - Se utilizó el manual para consultar el uso de ciertas funciones como *random_between()*, *random_member()*, *ansi_format()*, etc.