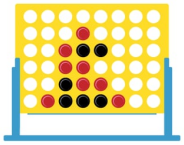
Práctica 1:
Conocimiento y Razonamiento Automatizado

Escuela Politécnica Superior, Alcalá de Henares(Madrid) España

---

Conecta 4.



Darius Dumitras Tamas
Luis Enrique Casado Alejandre
Daniel Verduras Gallego

Reparto de tareas:

El reparto de tareas básicas es el siguiente, no desgranamos más por que sería introducir demasiado detalle, no obstante cabe destacar que dentro de las tareas ha habido un trabajo entre múltiples personas, el nombre indicado después de la flecha indica quién ha implementado una mayor parte del código.

1. Implementar tablero. → Todos
2. Implementar introducir ficha → Daniel Verduras Gallego
3. Implementar los turnos por cada jugador. → Todos
4. Implementar victoria. → Darius Dumitras Tamas
5. Implementar empate. → Luis Enrique Casado Alejandre
6. Implementar el predicado jugar y jugando → Todos
7. Implementar inteligencia random.→ Luis Enrique Casado Alejandre
8. Implementar inteligencia artificial lista → Daniel Verduras Gallego
9. Implementar mejora IA vs IA → Daniel Verduras Gallego
10. Escribir memoria: → Darius Dumitras Tamas

Grado de cumplimiento de los requisitos:

Objetivos de la práctica:

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

Grado de cumplimentación: Completo, desde el tablero creado con lista, hasta la comprobación del ganador de la partida se ha hecho con funciones de listas. El ejemplo insertado debajo es un fragmento de nuestro código que comprueba si el jugador con la ficha “P” y teniendo el tablero “L” ha ganado, dicha búsqueda se realiza mediante la función específica de listas ***append***[1].

```
ganador(P, L) :- transpose(L, L1),
                 append(_, [C|_], L1),
                 append(_, [P,P,P,P|_], C).
```

2. El tablero de juego puede representarse internamente como una lista de listas o como una lista única de longitud 6x7. Esto queda a elección del alumno.

[1] https://www.swi-prolog.org/pldoc/man?predicate=append/3

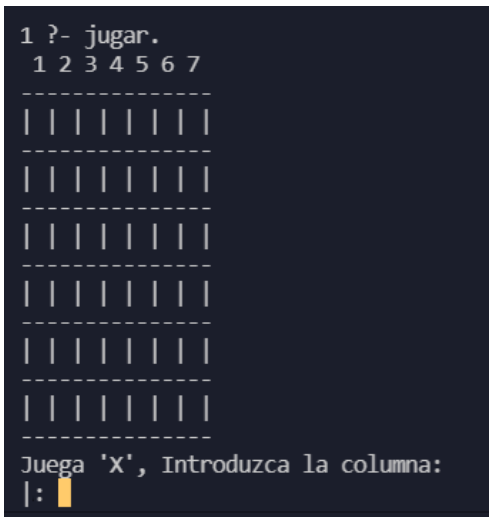
Grado de cumplimentación: Completo, nuestra elección fue crear una lista de listas. El tamaño total también ha sido fijo, 6x7. El código de la creación del tablero es el que se inserta a continuación.

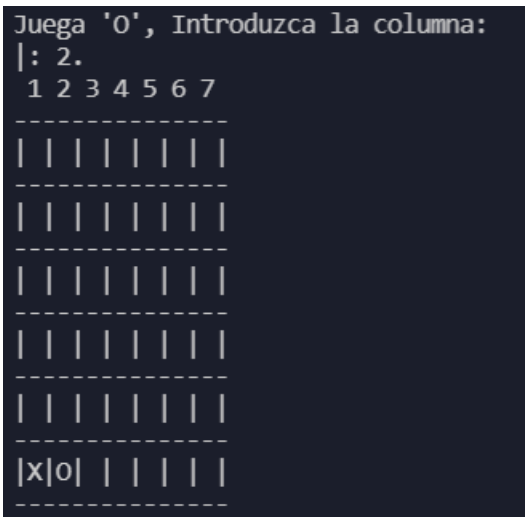
```
   lista_repe(1,X,[X]).
   lista_repe(N,X,[X|L]):- N1 is N-1, lista_repe(N1,X,L).

   generar_tablero_inicial(L) :- lista_repe(7, ' ', L1), lista_repe(6,L1,L), !.

   %%imprime los elementos de una lista
   escribir_lista([]).
   escribir_lista([X|Y]) :- write(X), escribir_lista(Y).

   %%imprime una fila con el formato: “ | elemento |... ”
   escribir_lista_con_barra([]).
   escribir_lista_con_barra([X|Y]):- write(X), write('|'),
escribir_lista_con_barra(Y).

   % imprime el n de columna en la cabecera
   escribir_tablero(L):- escribir_lista([' ', 1,' ', 2,' ', 3,' ', 4,' ', 5,' ',
6, ' ', 7]), nl, escribir_tablero1(L).

escribir_tablero1([]):- lista_repe(15,'-',L1), write(''), escribir_lista(L1), nl.

escribir_tablero1([X|L]):- lista_repe(15,'-',L1), write(''), escribir_lista(L1),
nl, write('|'), escribir_lista_con_barra(X), nl, escribir_tablero1(L).
```

3. Al principio, el jugador verá un tablero en el que en todos los lugares del mismo están vacíos (véase fichero EjemploConecta4.txt en la Blackboard de la asignatura). Esto exige implementar un predicado que escriba en pantalla un tablero 6x7 que se irá actualizando según el desarrollo del juego.

Grado de cumplimentación: Completo, en cada turno se actualiza el tablero después de cada movimiento, el tablero inicial también es idéntico al ejemplo proporcionado en EjemploConecta4.txt.

Las capturas de a continuación muestran cómo se imprime la pantalla.





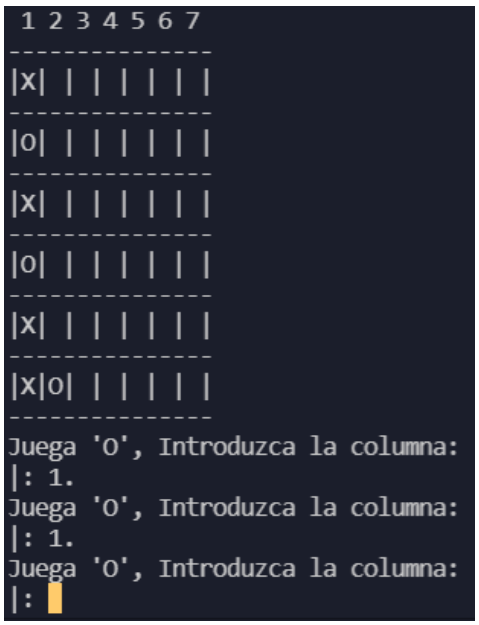
4. Hay que definir un predicado de aridad cero (jugar) que incorpore la preparación del juego y que, además, llame al predicado (jugando) que se encargue de gestionar el desarrollo del juego. Este predicado jugando, entre otras cosas, es el que se ocupará de preguntar al jugador por el número de columna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

Grado de cumplimentación: Completo, las capturas de pantalla de la pregunta anterior muestran tanto quién juega y le pide que introduzca una columna. Debajo se adjuntan nuestros predicados de jugar y jugando, en el que vemos cómo se llama a pedir_input para obtener la columna.

```
jugar:- generar_tablero_inicial(L), escribir_tablero(L), jugando('X', L), !.

jugando('X', L) :- ganador('O', L), write('Gana jugador 2').

jugando('O', L) :- ganador('X', L), write('Gana jugador 1').

jugando(_, L) :- \+ nocompleto(L), write('Empate').

jugando('X', L) :- repeat,write("Elige columna el jugador X"),pedir_input(C,
'X'), jugar_columna('X', C, L, L2),!, escribir_tablero(L2), jugando('O', L2).

jugando('O', L) :- repeat,write("Elige columna el jugador O"),pedir_input(C,
'O'), jugar_columna('O', C, L, L2),!,escribir_tablero(L2), jugando('X', L2).
```

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.

Grado de cumplimentación: Completo, en caso de que el jugador no pueda insertar en la columna ya que ésta está llena, se le pedirá de nuevo una nueva columna para insertar una ficha, en caso de que todo el tablero esté lleno, saldrá un mensaje que indicará: “Empate”.



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.

Grado de cumplimentación: Completo, en cualquiera de las 4 posibles geometrías que resulten victoriosas, la máquina lo detecta bien, a continuación se encuentran 4 capturas de pantalla que lo demuestran y el código fuente.

Código fuente:

```
% comprobacion de las columnas
ganador(P, L) :- append(_, [C|_], L),
                 append(_, [P,P,P,P|_], C).

% comprobacion de las filas
ganador(P, L) :- transpose(L, L1),
                 append(_, [C|_], L1),
                 append(_, [P,P,P,P|_], C).
```

```
% comprobando la diagonal de la forma \
ganador(P, L) :- append(_,[C1,C2,C3,C4|_],L),
                 append(I1, [P|_], C1),
                 append(I2, [P|_], C2),
                 append(I3, [P|_], C3), append(I4, [P|_], 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.

% comprobando la diagonal de la forma /
ganador(P, L) :- append(_,[C1,C2,C3,C4|_],L),
                 append(I1, [P|_], C1),
                 append(I2, [P|_], C2),
                 append(I3, [P|_], C3),
                 append(I4, [P|_], 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.

% código necesario para la comprobación de las filas:
transpose([[]|_], []) :- !.
transpose([[I|Is]|Rs], [Col|MT]) :- first_column([[I|Is]|Rs],
                                    Col,
                                    [Is|NRs]),
                                    transpose([Is|NRs], MT).
                                    first_column([], [], []).

first_column([[]|_], [], []).
first_column([[I|Is]|Rs], [I|Col], [Is|Rest]) :- first_column(Rs, Col, Rest).
```

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

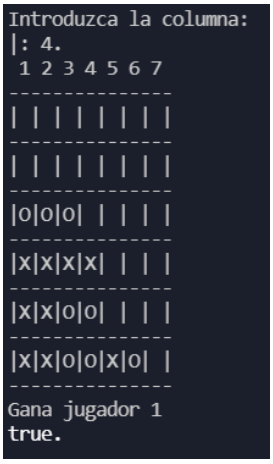
Grado de cumplimentación: Completo, se pueden observar las capturas de pantalla anteriores, donde jugador 1 = X y jugador 2 = O.

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 columna al azar en su turno. b. Estrategia avanzada: dotando de cierta inteligencia al proceso de toma de decisiones del ordenador.

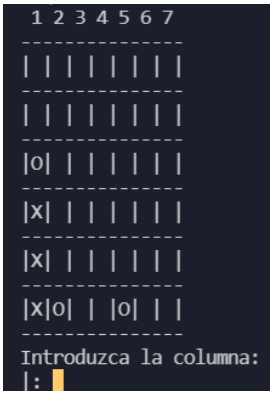
Grado de cumplimentación: Completo, la estrategia simple realiza movimientos aleatorios.

La estrategia avanzada realiza la siguiente secuencia de movimientos:
Primero intenta ganar, y en caso de que pueda, gana, si no puede ganar pero el rival sí que puede hacerlo, la máquina bloquea el movimiento para que no pueda ganar, mediante el uso de la función ***findall***[2].
Y en caso de que no existan dichos movimientos, se realiza un movimiento aleatorio,

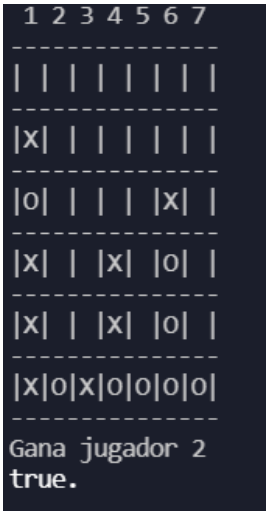
[2] https://www.swi-prolog.org/pldoc/man?predicate=findall/3



Ejemplo de una partida en la que la IA perdió.



Ejemplo de bloqueo de una casilla para impedir al jugador ganar.



Ejemplo en el que la máquina ha ganado.

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

Grado de cumplimentación: Completo, los nombres de los ficheros son: conecta4Aleatorio (Primera estrategia, movimiento aleatorio).
conecta4IA (Segunda estrategia, movimientos más óptimos).

Mejoras realizadas:

Hemos realizado, en el fichero llamado conecta4IAvsIA.pl la mejora

tenemos un predicado un poco diferente, que se llama inicio en vez de jugar, y se inicia como inicio luego se preguntará cuántas partidas se jugarán, al final de la ejecución, se muestra por pantalla un resumen de las partidas jugadas con las victorias.

```
Partidas ganadas por la IA lista: 17

Partidas ganadas por la IA random: 3
```

Durante las ejecuciones también se imprime el tablero con los movimientos que cada IA ha realizado en cada turno.

Errores y/o aspectos no implementados:

En cuanto a funcionalidad básica no implementada no existe nada no implementado.
Sin embargo, en cuanto a mejoras no implementadas, hemos optado por no implementar la mejora que especifica que el tamaño del tablero puede ser variable, y cuando eso suceda el número de fichas alineadas necesarias para la victoria cambian también.

Fuentes consultadas:

1. https://stackoverflow.com/questions/25467090/how-to-run-swi-prolog-from-the-command-line
2. https://stackoverflow.com/questions/5807455/matrix-operations-prolog
3. https://www.swi-prolog.org/pldoc/doc_for?object=manual
4. https://github.com/zachvav/connect4-prolog/blob/master/connect4.pl
5. https://github.com/rvinas/connect-4-prolog/blob/master/connect4.pl