

PL1 Conecta4

CONOCIMIENTO Y RAZONAMIENTO AUTOMATIZADO



17 de marzo de 2022

Grupo de laboratorio a3

Eduardo Ruiz Sabajanes, David Martinez Gutierrez, Jesús Palomino Abreu

# REPARTO DE TAREAS

El reparto de tareas ha sido el siguiente:

https://tohtml.com/sicstusProlog/

Semana 24 febrero:

Creación del tablero y de la interfaz

Semana 3 Marzo:

Insertar fichas, condiciones de victoria

Semana 10 Marzo:

Creación IA y mejoras generales

Semana 17 Marzo:

IA difícil, estadísticas, colores

# 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()*.

## EL TABLERO PUEDE REPRESENTARSE COMO UNA LISTA DE LISTAS

Como hemos visto en el apartado anterior, hemos decidido representar el tablero como una lista de listas, representando cada una de estas una columna.

## MOSTRAR TABLERO VACÍO AL COMIENZO

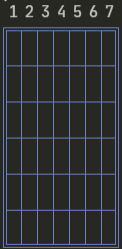
Como se puede ver más arriba, a la hora de rellenar cada columna introducimos el carácter espacio, por lo que, inicialmente, nuestro tablero vacío es un tablero lleno de espacios.

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.

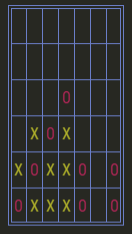
Seguidamente se llama a iShow(), encargada de mostrar la tabla. Se itera tantas veces como filas tengamos, y en cada iteración llama primero a dashLine() y luego a showLine().

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 el player 1 y el 2, llamando al predicado almacenado en NextTurn.

De esta forma vamos alternando entre el predicado del jugador 1 y el 2 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 como 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 !).