

DOMINÓ

Diego Villalvazo

José Sánchez

Sebastián Aranda

Objetivo




Implementación del Dominó

```
: -dynamic numeros/1.  
: -dynamic desconocidas/1.  
: -dynamic mano/1.  
: -dynamic turno/1.  
: -dynamic noTiene/1.  
: -dynamic tablero/1.  
: -dynamic extremoDerecho/1.  
: -dynamic extremoIzquierdo/1.  
: -dynamic pozo/1.  
: -dynamic posibles/1.
```

Main

```
2 ?- main.  
Ingresa las 7 fichas iniciales.  
|: [4,3].  
|: [2,1].  
|: [6,6].  
|: [4,1].  
|: [5,2].  
|: [1,0].  
|: [2,2].  
|: fin.  
¿Quién tiene el primer movimiento? yo/el  
|: █
```

Primer movimiento, tiro del oponente y nuestro tiro

```
¿Quién tiene el primer movimiento? yo/el
|: yo.
¿Cuál es la primera ficha que tiro?
|: [6,6].
¿El oponente tiró alguna ficha? si/no
|: si.
¿Qué ficha tiró el oponente?
|: [6,2].
¿De qué lado del tablero tiró el oponente? d/i
|: 1.
Se tiro del lado izquierdo.
----- [2,1]
¿El oponente tiró alguna ficha? si/no
|: 
```

Cuando el oponente roba del pozo

¿El oponente tiró alguna ficha? si/no

|: no.

¿Cuántas fichas tomó del pozo?

|: 4.

¿El oponente tiró alguna ficha? si/no

|: si.

¿Qué ficha tiró el oponente?

|: [1,1].

¿De qué lado del tablero tiró el oponente? d/i

|: 1.

Se tiro del lado izquierdo.

----- [4,1]

¿El oponente tiró alguna ficha? si/no

|: 

Cuando nosotros robamos del pozo

¿El oponente tiró alguna ficha? si/no

|: si.

¿Qué ficha tiró el oponente?

|: [0,0].

¿De qué lado del tablero tiró el oponente? d/i

|: 1.

Dame la ficha que robo.

|: 

Heurística

¿Cómo asignarle valor a un estado de juego?

Haciendo que el rival tenga que pasar.

- Recordar las fichas con las que ha pasado el rival.
- Estimar con qué fichas es más probable que pase.

Rival pasó

23 `noTiene([]).`

```
/**
 * Regla que evalua si el rival no tiene un número (la lista 'noTiene' registra
 * cuando el rival toma del pozo o pasa)
 * **/
rivalPaso(_, [], 0):- !.
rivalPaso(RivalPaso, [H|T], Ans):-
    member(H, RivalPaso),
    rivalPaso(RivalPaso, T, Resp),
    Ans is 2 + Resp, !.
rivalPaso(RivalPaso, [_|T], Ans):-
    rivalPaso(RivalPaso, T, Ans).
```

Estimación

```
21 pozo(14). /*Al inicio de cada juego de 1v1, el pozo empieza en 14 fichas. */
```

```
25 numeros([8,8,8,8,8,8,8]).
```

```
/* Regla que estima la posibilidad de que el rival no tenga un número determinado,  
 * recibe el número de fichas desconocidas totales. Utilizar para generar la estimación  
 * la lista que guarda cuantas fichas de cada grupo aún se desconocen.  
 */
```

```
estimacion(Desconocidas, NumPozo, QuedanNum, Estimacion):-  
    (NumPozo = 0) -> Estimacion is 0;  
    (NumPozo \= 0) -> Estimacion is (1-(QuedanNum/Desconocidas)).
```

Heurística

```
/**  
 * Regla que asigna un peso a un nodo determinado dependiendo del estado actual del  
 * juego, para la determinación del peso se consideran las fichas que el sistema  
 * desconoce, el número de fichas en el pozo, cuántas fichas compatibles con el  
 * tablero quedan todavía y las fichas que sabemos que el rival no tiene por que  
 * ha pasado.  
 **/  
pesoNodo(Desconocidas, NumPozo, [H|T], Peso):-  
    noTiene(N),  
    numeros(Nums),  
    nth0(H,Nums,L1),  
    nth0(T,Nums,L2),  
    estimacion(Desconocidas, NumPozo, L1, E1),  
    estimacion(Desconocidas, NumPozo, L2, E2),  
    Estimacion is E1 + E2,  
    rivalPaso(N, [H|T], NoTiene),  
    Peso is Estimacion + NoTiene.
```

Alfa Beta

```
/**
 * Regla que implementa la búsqueda alfa beta con poda, recibe el turno del jugador
 * la profundidad deseada, la posición del juego (tablero), las cotas alfa y beta, una
 * ficha compatible y regresa el peso si es que llegó a la profundidad deseada.
 * **/
alfa_beta(Player, Profundidad, Posicion, Alpha, Beta, Ficha, Peso) :-
    Profundidad > 0,
    compatibles(Player, Posicion, Movs),
    Alpha1 is -Beta,
    Beta1 is -Alpha,
    NuevaProf is Profundidad-1,
    evalua(Player, Movs, Posicion, NuevaProf, Alpha1, Beta1, nil, (Ficha, Peso)).
alfa_beta(_Player, 0, Ficha, _Alpha, _Beta, _SinMov, Peso) :-
    desconocidas(Desc),
    length(Desc, NumDesc),
    pozo(P),
    pesoNodo(NumDesc, P, Ficha, Peso).
```

Reglas auxiliares para alfa-beta: Evalúa

```
/**
 * Regla que evalúa el turno de un jugador con sus fichas disponibles, realiza una llamada
 * recursiva a alfa beta para evaluar cada una de las fichas disponibles de la mano del jugador,
 * una vez obtenido el peso de cada nodo hoja, llama a la función poda que se encarga de evaluar.
 * **/

evalua(Jugador1, [Ficha|Resto], Posicion, Profundidad, Alpha, Beta, Record, Mejor) :-
    %compatibles(Jugador1, Ficha, [Movs|_]),
    turno(Jugador1, Jugador2),
    alfa_beta(Jugador2, Profundidad, Ficha, Alpha, Beta, _OtroMov, Peso),
    Peso1 is -Peso,
    poda(Jugador1, Ficha, Peso1, Profundidad, Alpha, Beta, Resto, Posicion, Record, Mejor).
evalua(_Player, [], _Posicion, _D, Alpha, _Beta, Move, (Move, Alpha)).
```

Reglas auxiliares para alfa-beta: Poda

```
/**
 * Regla que evalúa el peso obtenido de cada nodo hoja con las cotas alfa y beta para decidir si
 * se detiene la ejecución (poda) e iniciar el retroceso, regresa la mejor ficha disponible.
 * **/
poda(_Jugador,Ficha,Peso,_Profundidad,_Alpha,Beta,_Resto,_Posicion,_Record,(Ficha,Peso)) :-
    Peso >= Beta,!.
poda(Jugador,Ficha,Peso,Profundidad,Alpha,Beta,Resto,Posicion,_Record,Mejor) :-
    Alpha < Peso,
    Peso < Beta,!,
    evalua(Jugador,Resto,Posicion,Profundidad,Peso,Beta,Ficha,Mejor).
poda(Jugador,_Ficha,Peso,Profundidad,Alpha,Beta,Resto,Posicion,Record,Mejor) :-
    Peso =< Alpha,!,
    evalua(Jugador,Resto,Posicion,Profundidad,Alpha,Beta,Record,Mejor).
```

Demostración del programa