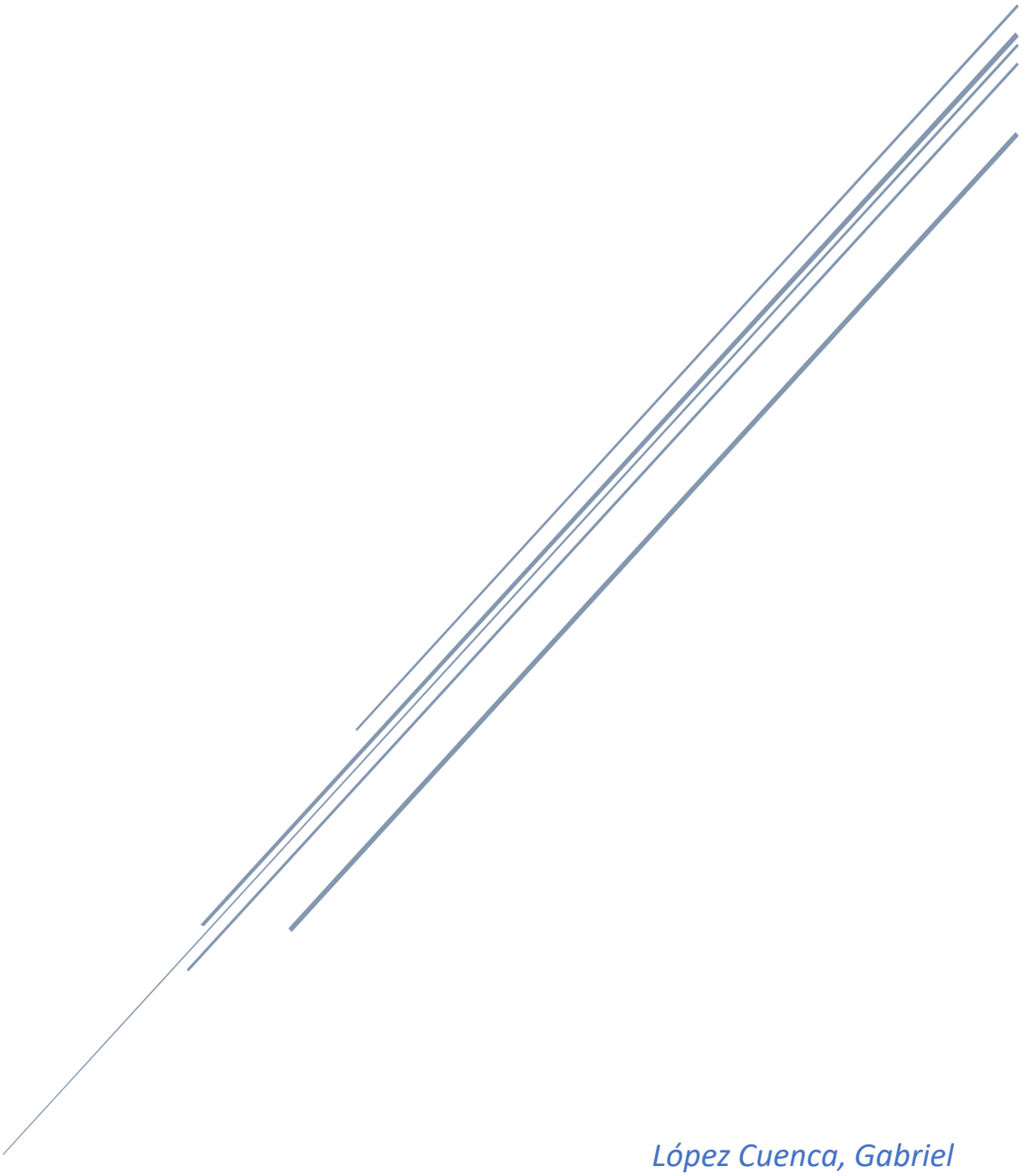
PRÁCTICA 2 LABORATORIO

INTELIGENCIA ARTIFICIAL

Universidad de Alcalá



López Cuenca, Gabriel
Sanz Sacristán, Sergio
Zamorano Ortega, Álvaro

Índice

FORMULACIÓN DEL PROBLEMA

Objetivo: realizar un programa en Racket con:

- Entradas: definir un bloque prismático de tres dimensiones, y definir un corte por cada turno del jugador, de una de las tres dimensiones, siempre que la longitud del corte de la dimensión seleccionada sea menor que el tamaño en ese momento de dicha dimensión.
- Salida: corte realizado por el ordenador y aviso de la finalización del juego, cuando el bloque prismático sea de dimensiones 1x1x1.

Entrada

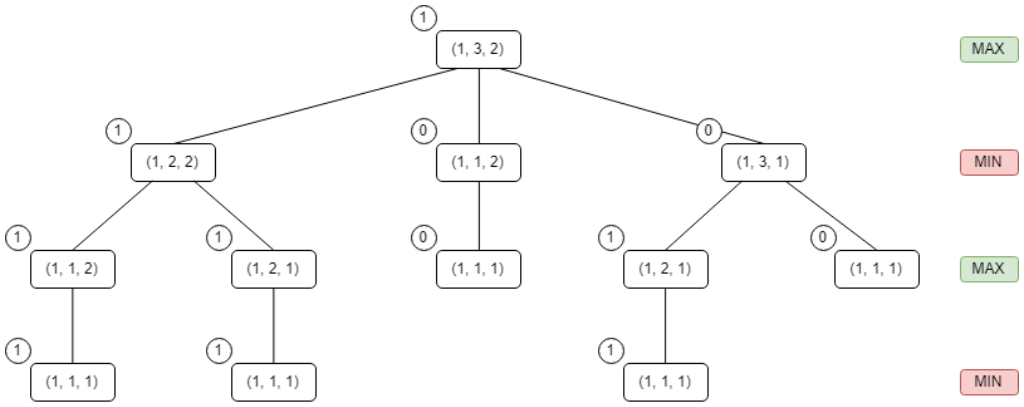
- Al comenzar la partida cuando se pide el tamaño del bloque prismático en cual se deben dar tres dimensiones para conformarlo. Estas tres dimensiones se deben dar por separado y positivas distintas de cero.
- Cuando es el turno del jugador, primero se introduce cual es la dimensión que se desea cortar, y posteriormente se debe indicar cual es la cantidad que se desea cortar. Siempre debe quedar un número positivo mayor o igual que uno tras el corte para que sea correcto.

Representación de la información

- Estado inicial: Lista (X, Y, Z) → X, Y, Z dimensiones del bloque prismático.
- Dimensión para recortar: {0, 1, 2} → Ancho, Alto o Largo
- Longitud que recortar: número natural menor que la longitud previa al corte de la dimensión seleccionada para el mismo.
- X, Y, Z ∈ Naturales
- Estado final: Lista (X, Y, Z) → X = Y = Z = 1

Operadores

El problema se resolverá mediante los métodos de búsqueda **Minimax**, para seleccionar el mejor movimiento posible para ganar al jugador. Planteamos un ejemplo de cómo este algoritmo funcionaría en el caso de que nos encontremos con un bloque prismático de tamaño (1, 3, 2):



Como podemos observar en el ejemplo anterior, cada vez que el turno es del ordenador, se realiza un árbol con todos los movimientos posibles partiendo de la raíz, la cual será la correspondiente a las dimensiones en ese momento del bloque prismático. Y se selecciona la mejor opción ganadora para el ordenador.

FUNCIONES AUXILIARES

prisma

- Retorna una lista con los tres números que recibe por parámetros, simulando las coordenadas de un prisma.

```
(define (prisma a b c) (list a b c))
```

```
> (prisma 1 2 3)
'(1 2 3)
```

prisma_fin?

- Retorna un booleano indicando si el prisma recibido es un prisma final, es decir, un prisma en el que no se pueden realizar cortes, del tipo (1 1 1).

```
(define (prisma_fin? x)
 (if (empty? x) #t (if (= (car x) 1) (prisma_fin? (cdr x)) #f)))
```

```
> (prisma_fin? (prisma 1 1 1))
#t
> (prisma_fin? (prisma 2 3 1))
#f
```

prisma_positivo?

- Retorna un booleano indicando si el prisma recibido contiene alguna coordenada negativa o igual a cero, lo cual no sería real.

```
(define (prisma_positivo? x)
 (if (empty? x) #t (if (> (car x) 0) (prisma_positivo? (cdr x)) #f)))
```

```
> (prisma_positivo? (prisma -1 2 3))
#f
> (prisma_positivo? (prisma 1 2 3))
#t
```

prisma_valido?

- Retorna un booleano indicando si el prisma recibido no es un prisma fin, y, además, es un prisma positivo.

```
(define (prisma_valido? prisma)
 (if (and (not (prisma_fin? prisma)) (prisma_positivo? prisma)) #t #f))
```

```
> (prisma_valido? (prisma 1 1 1))
#f
> (prisma_valido? (prisma -1 2 3))
#f
> (prisma_valido? (prisma 1 2 3))
#t
```

elemento_l

- Dada una lista y una posición (desde 1), devuelve elemento que se encuentre en dicha posición.

```
(define (elemento_l lista indice)
 (list-ref lista (- indice 1)))
```

```
> (elemento_l '(1 2 3) 2)
2
```

generarRandom

- Dado un límite, se genera un número aleatorio hasta él menos uno.

```
(define (generarRandom limite) (random limite))
```

```
> (generarRandom 5)
2
```

min

- Pasándole dos números, devuelve el menor de ellos.

```
(define (min n m) (if (>= m n) n m))
```

```
> (min 2 3)
2
```

max

- Pasándole dos números, devuelve el mayor de ellos.

```
(define (max n m) (if (>= m n) m n))
```

```
> (max 2 3)
3
```

FUNCIONES PARA REALIZAR CORTES

cortar_ancho

- Dado un prisma y un valor, devolverá un nuevo prisma con la primera de las coordenadas disminuida tanto como dicho valor.

```
(define (cortar_ancho x y) (prisma (- (elemento_l x 1) y) (elemento_l x 2)(elemento_l x 3)))
```

```
> (cortar_ancho '(3 2 3) 2)
'(1 2 3)
```

jugarAncho

- Dado un prisma, si el primer valor (anchura) es mayor que 1 (se puede realizar corte) se llama a corteAnchura, en caso contrario, se indica que esa dimensión no se puede modificar y se vuelve a pedir otra jugada.

```
(define (jugarAncho prisma)
 (if (> (elemento_l prisma 1) 1)
    (begin
     (display "\nElija un numero de corte \n")
     (corteAnchura prisma (read)))
    (begin
     (display "No se puede hacer el corte")
     (pedirJugada prisma))))
```

```
> (jugarAncho (prisma 1 2 3))
No se puede hacer el corte
Seleccione [1/2/3]-[Ancho/Alto/Largo]

> (jugarAncho (prisma 2 2 3))

Elija un numero de corte
1
'(1 2 3)
```

corteAnchura

- Recibe un prisma y un valor para el corte en anchura, es decir, de la primera coordenada. Si el valor es mayor o igual que esta, o cero, el corte no se permite y se vuelve a llamar a jugarAncho, en caso contrario, devuelve el nuevo prisma.

```
(define (corteAnchura prisma n)
 (if (or (>= n (elemento_l prisma 1)) (= n 0))
    (begin
     (display "Corte en anchura no posible\n")
     (jugarAncho prisma))
    (cortar_ancho prisma n)))
```

```
> (corteAnchura (prisma 2 2 3) 2)
Corte en anchura no posible

Elija un numero de corte
1
'(1 2 3)
> (corteAnchura (prisma 2 2 3) 1)
'(1 2 3)
```

listaCortesAncho

- Realiza una llamada a una función auxiliar.

```
(define (listaCortesAncho prisma) (listaCortesAncho_aux prisma 1))
```

listaCortesAncho_aux

- Genera una lista con todos los prismas resultantes de ir disminuyendo en 1 la primera de las coordenadas hasta que esta sea igual a 1.

```
(define (listaCortesAncho_aux prisma n)
 (if (< n (elemento_l prisma 1))
    (cons (cortar_ancho prisma n) (listaCortesAncho_aux prisma (+ n 1)))
    '()))
```

```
> (listaCortesAncho_aux (prisma 5 3 2) 1)
'((4 3 2) (3 3 2) (2 3 2) (1 3 2))
```

El resto de los cortes en las demás coordenadas se realiza de igual forma. Se ha considerado la altura del prisma a la segunda coordenada, y, por tanto, su profundidad es la tercera de ellas.

listaJugadas

- Genera una lista con todos los prismas resultantes de ir disminuyendo en 1 todas las coordenadas del prisma hasta que cada una de ellas sea 1. Lo único que hace es realizar llamadas al resto de funciones (listaCortesAncho, listaCortesAlto, listaCortesLargo).

```
(define (listaJugadas prisma)

 (append (append (listaCortesAncho prisma) (listaCortesAlto prisma)) (listaCortesLargo prisma)))
```

```
> (listaJugadas (prisma 2 3 4))
'((1 3 4) (2 2 4) (2 1 4) (2 3 3) (2 3 2) (2 3 1))
```

FUNCIONES PARA MINIMAX

nodoMinMax

- Dada una lista de tres números que representa un prisma y su nivel dentro del árbol, devuelve un 1 o un 0 en función de si por ese nodo del árbol existe un camino por el que se gana (1) o se pierde (0). Esta función es recursiva, llamando a las funciones *calcNodoMax* y *calcNodoMin* hasta llegar al prisma fin '(1 1 1), y en función de si este se encuentra en el nivel máximo o mínimo del árbol, devolverá un 1 (existe un camino por el que se gana) o un 0 (no existe un camino por el que se gane).

```
(define (nodoMinMax nodo nivel)
 (if (prisma_fin? nodo)
    (if (= nivel 0) 0 1)
    (let ([listaJugadas (listaJugadas nodo)])
     (if (= nivel 0)
        (calcNodoMax listaJugadas)
        (calcNodoMax listaJugadas)))))
```

```
> (nodoMinMax '(1 2 2) 1)
1
```

calcNodoMax

- Dada una lista de las jugadas posibles de un prisma cualquiera que se encontraba en un nivel max, calcula el máximo de los valores posibles de cada prisma de la lista llamando recursivamente consigo misma y a la función *nodoMinMax* de cada prisma. En caso de no haber jugadas disponibles (lista de jugadas es null), se devuelve un 0 siendo este el máximo.

```
(define (calcNodoMax listaJugadas)
 (if (= (length listaJugadas) 0)
    0
    (max (max 0 (nodoMinMax (car listaJugadas) 1)) (calcNodoMax (cdr listaJugadas)))))
```

```
> (listaJugadas '(1 2 1))
'((1 1 1))
> (calcNodoMax (listaJugadas '(1 2 1)))
1
```

calcNodoMin

- Dada una lista de las jugadas posibles de un prisma cualquiera que se encontraba en un nivel min, calcula el mínimo de los valores posibles de cada prisma de la lista llamando recursivamente consigo misma y a la función *nodoMinMax* de cada prisma. En caso de no haber jugadas disponibles (lista de jugadas es null), se devuelve un 1 siendo este el mínimo.

```
(define (calcNodoMin listaJugadas)
 (if (= (length listaJugadas) 0)
   1
   (min (min 1 (nodoMinMax (car listaJugadas) 0)) (calcNodoMin (cdr listaJugadas)))))
```

```
> (listaJugadas '(1 2 2))
'((1 1 2) (1 2 1))
> (calcNodoMin (listaJugadas '(1 2 2)))
1
```

juegaMaquina_aux

- Dada una lista de las jugadas posibles de un prisma raíz que se encuentra en el nivel max, se calcula cuál es la jugada (prisma) ganadora. Para ello, llama a la función *nodoMinMax* de cada jugada posible, y si ésta devuelve un 1 (por ese nodo pasa un camino ganado), la función devolverá dicho prisma. En cambio, devolverá un -1 si ninguna de las jugadas es ganadora. Dicha función retorna la primera jugada ganadora de la lista de posibles jugadas dada.

```
(define (juegaMaquina_aux listaJugadas)
 (if (= (length listaJugadas) 0) -1
   (if (= (nodoMinMax (car listaJugadas) 1) 1)
     (car listaJugadas)
     (juegaMaquina_aux (cdr listaJugadas)))))
```

```
> (listaJugadas '(1 3 2))
'((1 2 2) (1 1 2) (1 3 1))
> (juegaMaquina_aux (listaJugadas '(1 3 2)))
'(1 2 2)
```

juegaMaquina

- Dada un prisma del nodo raíz del árbol, calcula la mejor jugada llamando a la función juegaMaquina_aux con las jugadas posibles ya calculadas de este prisma.

Si al llamar a esta función no se devuelve una jugada, sino que se devuelve un número (-1), la función una jugada aleatoria entre todas las posibles.

```
(define (juegaMaquina prisma)
 (begin
  (display "\n*** Juega la maquina ***\n")
  (let ([jugadasPosibles (listaJugadas prisma)])
   (let ([resultadoJuegaMaquina (juegaMaquina_aux jugadasPosibles)])
    (if (number? resultadoJuegaMaquina)
      (elemento_l jugadasPosibles (+ (generarRandom (length jugadasPosibles)) 1)) ;Si no hay ganadora -> aleatoria entre posibles
      resultadoJuegaMaquina)))))
```

```
> (juegaMaquina '(1 3 2))

*** Juega la maquina ***
'(1 2 2)
```

EJECUCIÓN

juego

- En esta función imprime por pantalla la bienvenida al juego y llamar a la función jugar, la cual comienza a gestionar la ejecución del juego.

```
(define (juego)
 (begin
  (display "\n*** BIENVENIDO AL JUEGO DE LA CANTERA ***")
  (display "\n  *** Gabriel - Sergio - Alvaro ***\n")
  (jugar)))
```

jugar

- Esta función en primer lugar llama a la función correspondiente para pedir las dimensiones del prisma con el que se comenzará el juego (pedirPrisma). Posteriormente a la función turno la cual nos dará que jugador comenzará primero, si el resultado es cero comenzará el jugador y llama a la función partida con el prisma correspondiente e indicando que juega el jugador. En caso contrario llama a la función partida indicando el prisma y que la jugada la realizará la máquina.

```
(define (jugar)
 (begin
  (let([prism (pedirPrisma)])
   (if (= (turno) 0)
     (begin
      (display "\n*** COMIENZA EL JUGADOR ***\n")
      (partida prism 0))
     (begin
      (display "\n*** COMIENZA LA MAQUINA ***\n")
      (partida prism 1))))))
```

pedirPrisma

- En esta función se pide al jugador que introduzca por pantalla cuáles serán las dimensiones del prisma con el que se realizará la partida. Se crea el prisma llamando a la función prisma introduciéndole como dimensiones los introducidos

por pantalla por el jugador. Además, se tiene en cuenta que estos valores sean correctos mediante la función prisma_valido?

```
(define (pedirPrisma)
 (begin
  (display "\nIntroduce las dimensiones del prisma\n")
  (let ([prism (prisma (read)(read)(read))])
   (if (prisma_valido? prism)
      prism
      (begin
       (display "* Dimensiones del prisma inválidas *\n")
       (pedirPrisma))))))
```

```
> (pedirPrisma)

Introduce las dimensiones del prisma
1 3 2
'(1 3 2)
```

turno

- En esta función se genera un número aleatorio, o el 0 o el 1, el cual servirá para saber quién jugará primero, si el jugador (0) o la máquina (1).

```
(define (turno) (generarRandom 2))
```

```
> (turno)
1
> (turno)
1
> (turno)
0
```

partida

- A esta función se le introducen como datos, el prisma inicial y el jugador que lleva a cabo la jugada. Primero se imprime por pantalla el prisma con el que se está trabajando en ese momento. Posteriormente, se comprueba si dicho prisma es el final del juego, si esto se cumple dependiendo del jugador del que sea el turno habrá ganado la maquina o el jugador. Si el prisma aun no es el final entonces llamaremos de nuevo a la función partida con el prisma modificado tras el corte correspondiente en ese turno.

```
(define (partida prisma jugador)
 (begin
  (display "\n*** Prisma actual ")
  (display prisma)
  (display " ***\n")
  (if (prisma_fin? prisma)
   (if (= jugador 0) (display "\n*** HAS PERDIDO ***\n") (display "\n*** HAS GANADO ***\n"))
   (if (= jugador 0)
      (partida (pedirJugada prisma) 1)
      (partida (juegaMaquina prisma) 0)))))
```

pedirJugada

- En esta función se pide por pantalla a que dimensión realizaremos el corte. El jugador selecciona la dimensión que desea introduciendo 1, 2 o 3. Si la dimensión seleccionada es el Ancho se llamará a la función jugarAncho, si selecciona Alto a la función jugarAlto y si selecciona Largo a jugarLargo.

```
(define (pedirJugada prisma)
 (begin
  (display "\nSeleccione [1/2/3]-[Ancho/Alto/Largo]\n");
  (let ([direccion (read)])
   (cond
    [(= direccion 1) (jugarAncho prisma)]
    [(= direccion 2) (jugarAlto prisma)]
    [(= direccion 3) (jugarLargo prisma)]
    [else (begin
          (display "\nNo ha seleccionado una opcion correcta\n")
          (pedirJugada prisma))])))))
```

```
> (pedirJugada '(1 3 2))

Seleccione [1/2/3]-[Ancho/Alto/Largo]
2

Elija un numero de corte
1
'(1 2 2)
```