Universidad Autónoma de Madrid

Inteligencia Artificial

práctica 3

Nombres	Daniel Tijerina González
	Suemy Inagaki Pinheiro Fagundes
Grupo	2363

Madrid, 03 de Abril de 2020

1 Función de Evaluación 1

1.1 Analise de los Requisitos

- la funcion recibe un player y un board
- debe retornar un número

1.2 Pseudocódigo

```
Inicio Funcion (player board)
    verifica la diferencia en la cantidad de casas
    malas el oponente y el player van a tener en el
    proximo movimiento.
    multiplica esa diferencia por 6

    verifica la diferencia en la cantidad de esquinas
    que el player y el oponente tienen en este board.
    multiplica esa diferencia por 9

    verifica la diferencia en la catidad de esquinas
    que el player y el oponente van a tener en los
    proximos posibles movimientos.
    multiplica esa diferencia por 4.1

    verifica la diferencia en la cantidad de movimientos
    posibles el player y el oponente van a tener.

    suma todos los valores arriba y retorna.
Fin Funcion
```

1.3 Definición de la Función

```
;*************************************************
; FUNCIONES AUXILIARES
;*************************************************

; funcion que recibe una funcion q y una lista l y remove
; los elementos que no satisfacen la funcion.

(defun filter (l q)
  (remove-if-not q l))

; funcion que recibe un board y retorna los esquinas

(defun corners (board)
```

```
  (list (aref board 11)
        (aref board 18)
        (aref board 81)
        (aref board 88)))

; funcion que recibe una lista y retorna los elementos
; que pertenecen a lista y que son esquinas

(defun get-corners (l)
  (length (filter l #'(lambda (x) (member x
    '(11 18 81 88))))))

; Funcion que recibe un board y retorna una lista con las
; casas que consideramos malas para el oponente.
; Son las casas de las bordas y las diagonales sin los esquinas.
; Así, debemos forzar el oponente a moverse para esas casas.

(defun sides-x (board)
  (mapcar #'(lambda (x) (aref board x))
  '(12 13 14 15 16 17
      82 83 84 85 86 87 78
      68 58 48 38 28 71 61
      51 41 31 21 22 33 44 55
      66 77 72 63 54 45 36 27)))


; la funcion recibe un board y una lista.
; la funcion busca en la lista las casas
; malas y retorna el numero de casas malas
; que hay na lista.

(defun num-sides-x (board lista)
  (if (null lista)
    0
    (if (find (car lista) (sides-x board))
      (+ 1 (num-sides-x board (cdr lista)))
      (num-sides-x board (cdr lista)))))


;*************************************************
; FUNCION PRINCIPAL
;*************************************************

(defun eval-fn (player board)
  (+
```

```
(* 6
  (-
    (num-sides-x board (legal-moves (opponent player) board))
    (num-sides-x board (legal-moves player board))))
(* 9
  (-
    (length (filter (corners board) #'(lambda (x) (eq x player))))
    (length (filter (corners board) #'(lambda (x) (eq x (opponent player)))))))
(* 4.1
    (-
      (get-corners (legal-moves player board))
      (get-corners (legal-moves (opponent player) board))))

   (-
      (length (legal-moves player board))
      (length (legal-moves (opponent player) board)))))
```

1.4 Comentarios

Consideramos importante que el oponente se quede con el máximo de casas malas posibles, sin que el jugador sea prejudicado con eso, por eso hemos calculado la diferencia entre el oponente y el jugador en este caso.

La condicion arriba permite que el oponente tenga menos posibilidades de ganar la partida pero pensamos que eso no es suficiente para que el jugador gane la partida. Para que eso ocurra, el jugador tendría que dominar los esquinas del board. Así que ponemos el mayor peso en esta diferencia, debido a su importancia.

Los pesos fueron puestos de acuerdo con los tests que realizamos. Sabiamos la orden de importancia y asi fuimos ponendolos y modificandolos de acuerdo con los resultados de los tests. Así, la mejor combinacion de pesos es 6, 9, 4.1 y 1, como ponemos en la funcion.

2 Función de Evaluación 2

2.1 Analise de los Requisitos

- la funcion recibe un player y un board
- debe retornar un número

2.2 Pseudocódigo

```
Inicio Funcion (player board)
    verifica la diferencia en la cantidad de casas
    malas el oponente y el player van a tener en el
    proximo movimiento.
    multiplica esa diferencia por 6
```

```
    verifica la diferencia en la cantidad de esquinas
    que el player y el oponente tienen en este board.
    multiplica esa diferencia por 9

    verifica la diferencia en la catidad de esquinas
    que el player y el oponente van a tener en los
    proximos posibles movimientos.
    multiplica esa diferencia por 4.1

    verifica la diferencia en la cantidad de movimientos
    posibles el player y el oponente van a tener.
    multiplica esta diferencia por 1.2

    suma todos los valores arriba y retorna.
Fin Funcion
```

2.3 Definición de la Función

```
;*********************************************
; FUNCIONES AUXILIARES
;*********************************************


; funcion que recibe una funcion q y una lista l y remove
; los elementos que no satisfacen la funcion.

(defun filter (l q)
  (remove-if-not q l))

; funcion que recibe un board y retorna los esquinas

(defun corners (board)
  (list (aref board 11)
        (aref board 18)
        (aref board 81)
        (aref board 88)))

; funcion que recibe una lista y retorna los elementos
; que pertenecen a lista y que son esquinas

(defun get-corners (l)
  (length (filter l #'(lambda (x) (member x
    '(11 18 81 88))))))

; Funcion que recibe un board y retorna una lista con las
; casas que consideramos malas para el oponente.
```

```
; Son las casas de las diagonales sin los esquinas y
; las casas cerca del esquina.
; Así, debemos forzar el oponente a moverse para esas casas.

(defun sides-x1 (board)
  (mapcar #'(lambda (x) (aref board x))
  '(12 21 22 82 71 72 87 78 77 28 17 27 33 44 55
      66 63 54 45 36 )))


; la funcion recibe un board y una lista.
; la funcion busca en la lista las casas
; malas y retorna el numero de casas malas
; que hay na lista.

(defun num-sides-x1 (board lista)
  (if (null lista)
    0
    (if (find (car lista) (sides-x1 board))
      (+ 1 (num-sides-x1 board (cdr lista)))
      (num-sides-x1 board (cdr lista)))))

;***********************************************
; FUNCION PRINCIPAL
;***********************************************

 (defun eval-fn (player board)
  (+
    (* 6
      (-
        (num-sides-x1 board (legal-moves (opponent player) board))
        (num-sides-x1 board (legal-moves player board))))
    (* 9
      (-
        (length (filter (corners board) #'(lambda (x) (eq x player))))
        (length (filter (corners board) #'(lambda (x) (eq x (opponent player)))))))
    (* 4.1
        (-
          (get-corners (legal-moves player board))
          (get-corners (legal-moves (opponent player) board))))

    (* 1.2
        (-
          (length (legal-moves player board))
          (length (legal-moves (opponent player) board))))
```

```
))
```

2.4 Comentarios

Los comentarios son los mismos para la primera heuristica presentada en esta memoria. La diferencia es que ahora las casas malas no incluyen los bordes.

El resultado obtenido está muy cerca del obtenido por la primera heuristica, así, consideramos importante ponerla aqui.

3 Función de Evaluación 3

- la funcion recibe un player y un board
- debe retornar un número

3.1 Pseudocódigo

```
Inicio Funcion (player board)
    verifica la diferencia en la cantidad de casas
    malas el oponente y el player van a tener en el
    proximo movimiento.
    multiplica esa diferencia por 6

    verifica la diferencia en la cantidad de esquinas
    que el player y el oponente tienen en este board.
    multiplica esa diferencia por 9

    verifica la diferencia en la catidad de esquinas
    que el player y el oponente van a tener en los
    proximos posibles movimientos.
    multiplica esa diferencia por 4.1

    verifica la diferencia en la cantidad de movimientos
    posibles el player y el oponente van a tener.

    suma todos los valores arriba y retorna.
Fin Funcion
```

3.2 Definición de la Función

```
;*************************************************
; FUNCIONES AUXILIARES
;*************************************************


; funcion que recibe una funcion q y una lista l y remove
```

```
; los elementos que no satisfacen la funcion.

(defun filter (l q)
  (remove-if-not q l))

; funcion que recibe un board y retorna los esquinas

(defun corners (board)
  (list (aref board 11)
        (aref board 18)
        (aref board 81)
        (aref board 88)))

; funcion que recibe una lista y retorna los elementos
; que pertenecen a lista y que son esquinas

(defun get-corners (l)
  (length (filter l #'(lambda (x) (member x
    '(11 18 81 88))))))

; Funcion que recibe un board y retorna una lista con las
; casas que consideramos malas para el oponente.
; Son las casas de las diagonales sin los esquinas y
; las casas cerca del esquina.
; Así, debemos forzar el oponente a moverse para esas casas.

(defun sides-x1 (board)
  (mapcar #'(lambda (x) (aref board x))
  '(12 21 22 82 71 72 87 78 77 28 17 27 33 44 55
      66 63 54 45 36 )))


; la funcion recibe un board y una lista.
; la funcion busca en la lista las casas
; malas y retorna el numero de casas malas
; que hay na lista.

(defun num-sides-x1 (board lista)
  (if (null lista)
    0
    (if (find (car lista) (sides-x1 board))
      (+ 1 (num-sides-x1 board (cdr lista)))
      (num-sides-x1 board (cdr lista)))))

;***********************************************
```

```
; FUNCION PRINCIPAL
;*********************************************

(defun eval-fn (player board)
  (+
    (* 6
      (-
        (num-sides-x1 board (legal-moves (opponent player) board))
        (num-sides-x1 board (legal-moves player board))))
    (* 9
      (-
        (length (filter (corners board) #'(lambda (x) (eq x player))))
        (length (filter (corners board) #'(lambda (x) (eq x (opponent player)))))))
    (* 4.1
        (-
          (get-corners (legal-moves player board))
          (get-corners (legal-moves (opponent player) board))))

    (-
          (length (legal-moves player board))
          (length (legal-moves (opponent player) board)))))
```

3.3 Comentarios

Los comentarios son los mismos para la segunda heuristica presentada en esta memoria. La diferencia es que hemos puesto un peso de 1.2 para la diferencia de los posibles movimientos.

El resultado obtenido está muy cerca de los obtenidos en las heuristicas 1 y 2.

4 Funcion de Evaluacion 4

4.1 Análise de los Requisitos

- la funcion recibe un player y un board
- debe retornar un número

4.2 Pseudocodigo

```
Inicio Funcion (player board)
    retorna la diferencia entre las cantidades de
    esquinas que el player y el oponente tienen
Fin Funcion
```

4.3 Definicion de la Funcion

```
(defun eval-fn (player board)
  (let ((corners (list (aref board 11) (aref board 18)
                       (aref board 81) (aref board 88))))
    (- (length (remove-if-not #'(lambda (x) (eq x player)) corners))
       (length (remove-if-not #'(lambda (x) (eq x (opponent player))) corners)))))
```

4.4 Comentarios

Es una heurística muy simple y no muy efectiva, ya que desconsidera muchas cosas que no podría desconsiderar.

5 Funcion de Evaluacion 5

5.1 Análise de los Requisitos

- la funcion recibe un player y un board
- debe retornar un número

5.2 Pseudocodigo

```
Inicio Funcion (player board)
    calcula la diferencia entre la cantidad de
    esquians  que cada jugador tiene.
    multiplica esa diferencia por 5

    calcula la diferencia entre la cantidade de
    esquina que cada jugador vay a tener de acuerdo
    con las posibles movimientos.

    suma los valores y retorna.
Fin Funcion
```

5.3 Definicion de la Funcion

```
;***********************************************
; FUNCIONES AUXILIARES
;***********************************************


; funcion que recibe una funcion q y una lista l y remove
; los elementos que no satisfacen la funcion.

(defun filter (l q)
```

```
  (remove-if-not q l))

; funcion que recibe un board y retorna los esquinas

(defun corners (board)
  (list (aref board 11)
        (aref board 18)
        (aref board 81)
        (aref board 88)))

; funcion que recibe una lista y retorna los elementos
; que pertenecen a lista y que son esquinas

(defun get-corners (l)
  (length (filter l #'(lambda (x) (member x
    '(11 18 81 88))))))

;*********************************************
; FUNCION PRINCIPAL
;*********************************************

(defun eval-fn (player board)
(+
  (-
   (* 5 (length (filter (corners board) #'(lambda (x) (eq x player)))))
   (* 5 (length (filter (corners board) #'(lambda (x) (eq x (opponent player)))))))
  (-
   (get-corners (legal-moves player board))
   (get-corners (legal-moves (opponent player) board)))))
```

5.4 Comentarios

Es una heuristica que lleva en consideracion la importancia de conquistar las esquinas. Así, atribuimos 5 a su peso y 1 para los posibles movimientos que envuelven conquistarla esquina.

6 Funcion de Evaluacion 6

6.1 Análise de los Requisitos

- la funcion recibe un player y un board
- debe retornar un número

6.2 Pseudocodigo

```
Inicio Funcion (player board)
```

```
    calcula la diferencia entre la cantidad de
    esquina que cada jugador tiene.
    multiplica esa diferencia por 5

    calcula la diferencia entre la cantidade de
    esquina que cada jugador vay a tener de acuerdo
    con las posibles movimientos.

    Calcula la diferencia entre la cantidad de piezas
    en la borda que cada jugador tiene
    multiplica esa diferencia por 2

    suma los valores y retorna.
Fin Funcion
```

6.3 Definicion de la Funcion

```
;***********************************************
; FUNCIONES AUXILIARES
;***********************************************


; funcion que recibe una funcion q y una lista l y remove
; los elementos que no satisfacen la funcion.

(defun filter (l q)
  (remove-if-not q l))

; funcion que recibe un board y retorna los esquinas

(defun corners (board)
  (list (aref board 11)
        (aref board 18)
        (aref board 81)
        (aref board 88)))

; funcion que recibe una lista y retorna los elementos
; que pertenecen a lista y que son esquinas

(defun get-corners (l)
  (length (filter l #'(lambda (x) (member x
    '(11 18 81 88))))))


(defun sides (board)
  (mapcar #'(lambda (x) (aref board x))
```

```
    '(13 14 15 16 31 41 51 61 38 48 58 68 83 84 85 86)))

;*********************************************
; FUNCION PRINCIPAL
;*********************************************

(defun eval-fn (player board)
(+
  (-
   (* 5 (length (filter (corners board) #'(lambda (x) (eq x player)))))
   (* 5 (length (filter (corners board) #'(lambda (x) (eq x (opponent player)))))))
  (-
   (get-corners (legal-moves player board))
   (get-corners (legal-moves (opponent player) board)))
  (-
   (* 2 (length (filter (sides board) #'(lambda (x) (eq x player)))))
   (* 2 (length (filter (sides board) #'(lambda (x) (eq x (opponent player))))))))
```

6.4 Comentarios

En esta heuristica intentamos darle más importancia a la cantidad de esquinas que el player tiene, pero tambien ponemos un peso mayor en la conquista de las bordas.

No era una mala heuristica pero no era suficiente para obtener victorias.

7 Funcion de Evaluacion 7

7.1 Análise de los Requisitos

- la funcion recibe un player y un board
- debe retornar un número

7.2 Pseudocodigo

```
Inicio Funcion (player board)
    calcula la diferencia entre la cantidad de
    esquina que cada jugador tiene.
    multiplica esa diferencia por 5

    calcula la diferencia entre la cantidade de
    esquinas que cada jugador vay a tener de acuerdo
    con los posibles movimientos.

    Calcula la diferencia entre la cantidad de piezas
    en la borda que cada jugador tiene
    multiplica esa diferencia por 2
```

```
    calcula la diferencia entre la cantidad de posibles
    movimientos entre los jugadores.
    multiplica la diferencia por 0.5

    suma los valores y retorna.
Fin Funcion
```

7.3 Definicion de la Funcion

```
;*****************************************************
; FUNCIONES AUXILIARES
;*****************************************************

; funcion que recibe una funcion q y una lista l y remove
; los elementos que no satisfacen la funcion.

(defun filter (l q)
  (remove-if-not q l))

; funcion que recibe un board y retorna los esquinas

(defun corners (board)
  (list (aref board 11)
        (aref board 18)
        (aref board 81)
        (aref board 88)))

; funcion que recibe una lista y retorna los elementos
; que pertenecen a lista y que son esquinas

(defun get-corners (l)
  (length (filter l #'(lambda (x) (member x
    '(11 18 81 88))))))


(defun sides (board)
  (mapcar #'(lambda (x) (aref board x))
    '(13 14 15 16 31 41 51 61 38 48 58 68 83 84 85 86)))

(defun get-sides (l)
  (length (filter l #'(lambda (x) (member x
    '(13 14 15 16 31 41 51 61 38 48 58 68 83 84 85 86))))))

;*****************************************************
; FUNCION PRINCIPAL
```

```
;*****************************************************

(defun eval-fn (player board)
(+
  (-
    (* 5 (length (filter (corners board) #'(lambda (x) (eq x player)))))
    (* 5 (length (filter (corners board) #'(lambda (x) (eq x (opponent player)))))))
  (-
          (get-corners (legal-moves player board))
          (get-corners (legal-moves (opponent player) board)))
  (-
    (* 2 (length (filter (sides board) #'(lambda (x) (eq x player)))))
    (* 2 (length (filter (sides board) #'(lambda (x) (eq x (opponent player)))))))
  (-
    (* 0.5 (get-sides (legal-moves player board)))
    (* 0.5 (get-sides (legal-moves (opponent player) board))))))
```

7.4 Comentarios

La diferencia de esta heuristica con la heuristica 6 es que hemos puesto la diferencia de posibles movimientos que el jugaor y el oponente tienen, con un peso pequeño.

Pero como en la 6, no era suficiente para obtener victorias.

8 Resultados Obtenidos

Testamos todas las heuristicas que implementamos junto con las heuristicas dadas (mobility, count-difference y random-strategy):

```
;*****************************
; TEST 1
;*****************************
COUNT-DIF            378.: ---  107. 137. 10.0 11.0 12.0 40.5 18.5 25.0 17.5
MOBILITY             440.: 93.5 ---  160. 13.5 10.0 10.5 56.5 22.0 36.5 37.5
RANDOM-STR           173.: 63.5 40.0 ---   3.5  2.0  9.0 22.5  6.0 13.0 13.0
EVAL-FN1             1368: 190. 187. 197. ---  102. 100. 170. 141. 142. 140.
EVAL-FN2             1359: 189. 190. 198. 98.0 ---  92.5 170. 149. 140. 134.
EVAL-FN3             1373: 188. 190. 191. 100. 108. ---  162. 142. 148. 145.
EVAL-FN4             753.: 160. 144. 178. 30.5 30.5 38.0 ---  43.5 65.5 64.0
EVAL-FN5             1109: 182. 178. 194. 59.0 51.5 58.0 157. ---  122. 109.
EVAL-FN6             1008: 175. 164. 187. 58.0 60.5 52.5 135. 78.5 ---  98.5
EVAL-FN7             1042: 183. 163. 187. 59.5 66.5 55.0 136. 91.0 102. ---

;*****************************
; TEST 2
;*****************************
```

```
EVAL-FN1            779.: ---  94.0 100. 169. 140. 145. 132.
EVAL-FN2            790.: 106. ---  106. 168. 135. 136. 140.
EVAL-FN3            796.: 100. 94.5 ---  176. 150. 138. 137.
EVAL-FN4            270.: 31.5 32.0 24.5 ---  36.0 78.5 68.0
EVAL-FN5            596.: 60.0 65.5 49.5 164. ---  132. 125.
EVAL-FN6            470.: 55.5 64.5 62.0 122. 68.5 ---  98.0
EVAL-FN7            500.: 68.5 60.0 63.0 132. 75.0 102. ---
```

Podemos ver en los resultados arriba que las heurísticas que presentan los mejores resultados son las heuristicas 1, 2 y 3, que intentan conquistar las esquinas y a forzar el oponente a hacer un mal movimiento.