JUEGO DEL MICHI

Introducción

Se desea implementar el juego llamado “Tres en línea” (Michi) en Common LISP usando la herramienta llamada LispWorks. El programa a desarrollarse no solamente permitirá efectuar las jugadas del humano sino que determinará las jugadas de la máquina mediante una estrategia adecuada. Asimismo indicará la estrategia usada para efectuar dichos movimientos.

Interfaz del juego

Los casilleros del tablero a usar en el Michi serán enumerados de la siguiente manera:

```
1 | 2 | 3
-----------
4 | 5 | 6
-----------
7 | 8 | 9
```

El tablero se representará como una lista formada por el símbolo `TABLERO` seguida de nueve números que describirán el contenido de cada posición. Un valor de 0 significa que la posición está vacía; un 1 significa que hay un círculo O, y un valor de 10 significa que hay un aspa X. Asumiremos que el oponente es O y que la computadora es X.

a) Definimos una función que llamaremos `CREAR-TABLERO` para crear un nuevo tablero.

```
(defun crear-tablero ()
        (list 'tablero 0 0 0 0 0 0 0 0 0))
```

b) Para imprimir el tablero definimos las funciones CONVERTIR-A-LETRAS, IMPRIMIR-FILA, e IMPRIMIR-TABLERO.

```
(defun convertir-a-letras (v)
  (cond ((equal v 1) "O")
        ((equal v 10) "X")
        (t " ")))

(defun imprimir-fila (x y z)
  (format t "~& ~A | ~A | ~A"
      (convertir-a-letras x)
      (convertir-a-letras y)
      (convertir-a-letras z)))

(defun imprimir-tablero (tablero)
  (format t "~%")
  (imprimir-fila
      (nth 1 tablero) (nth 2 tablero) (nth 3 tablero))
  (format t "~& -----------")
  (imprimir-fila
      (nth 4 tablero) (nth 5 tablero) (nth 6 tablero))
  (format t "~& -----------")
  (imprimir-fila
      (nth 7 tablero) (nth 8 tablero) (nth 9 tablero))
  (format t "~%~%"))
```

c) Definimos EFECTUAR-MOVIMIENTO para marcar un X o un O en el tablero, según sea el jugador que le toca jugar. JUGADOR puede asumir dos valores: *PC* y *OPONENTE*.

```
(defun efectuar-movimiento (jugador pos tablero)
   (setf (nth pos tablero) jugador)
   tablero)
```

Antes configuramos:

```
(setf *pc* 10 *oponente* 1)
```

Análisis de la configuración del juego

a) Las 8 maneras de completar una línea (3 horizontalmente, 3 verticalmente, y 2 en la diagonal) serán almacenadas en *TRIPLETAS*. A c/u de esas combinaciones la llamaremos "tripleta".

```
(setf *tripletas* '((1 2 3) (4 5 6) (7 8 9)
                    (1 4 7) (2 5 8) (3 6 9)
                    (1 5 9) (3 5 7)))
```

b) Definimos ahora SUMA-DE-TRIPLETA, que devuelve la suma de los números contenidos por una tripleta.

```
(defun suma-de-tripleta (tablero tripleta)
    (+ (nth (car tripleta) tablero)(nth (cadr tripleta) tablero)
    (nth (caddr tripleta) tablero))
)
```

c) Para observar las sumas de todas las tripletas definimos CALCULA-SUMAS.

```
(defun calcula-sumas (tablero)
 (mapcar #'(lambda (tripleta) (suma-de-tripleta tablero tripleta))
 *tripletas*))
```

Condición de parada

a) Definimos el predicado GANADOR-P para verificar una de las condiciones de parada: uno de los jugadores completó una línea

```
(defun ganador-p (tablero)
  (let ((sumas (calcula-sumas tablero)))
    (or (member (* 3 *pc*) sumas) (member (* 3 *oponente*) sumas)))
)
```

Inicio del juego

a) Definimos INICIAR-JUEGO que pregunta al usuario si desea comenzar el juego, y llama luego a la función MOVIMIENTO-PC o a MOVIMIENTO-OPONENTE, según la decisión tomada. Usa el predicado Y-OR-N-P para la condición del IF.

```
(defun iniciar-juego ()
  (if (y-or-n-p "¿Le gustaria comenzar el juego? ")
                (movimiento-oponente (crear-tablero))
                (movimiento-pc (crear-tablero)))
)
```

Jugadas del jugador humano

a) Definimos MOVIMIENTO-OPONENTE para efectuar el movimiento del humano. Además verifica con VALIDAR-MOVIMIENTO si movimiento es válido, actualiza el tablero y llama a MOVIMIENTO-PC, excepto que haya un ganador o un empate. Usa el predicado TABLERO-TOTAL-P para comprobar si el tablero está lleno.

```
(defun movimiento-oponente (tablero)
   (let* ((pos (validar-movimiento tablero))
          (nuevo-tablero (efectuar-movimiento *oponente* pos
tablero)))
     (imprimir-tablero nuevo-tablero)
     (cond ((ganador-p nuevo-tablero) (format t "~&Ud. gano!"))
           ((tablero-total-p nuevo-tablero) (format t "~&Empate."))
           (t (movimiento-pc nuevo-tablero))))
)

(defun validar-movimiento (tablero)
   (format t "~&Tu movimiento: ")
   (let ((pos (read)))
      (cond ((not (and (integerp pos) (<= 1 pos 9)))(format t
"~&Entrada invalida.") (validar-movimiento tablero))
            ((not (zerop (nth pos tablero))) (format t "~&Ese
espacio ya esta ocupado.") (validar-movimiento tablero))
            (t pos))
   )
 )

(defun tablero-total-p (tablero)
   (not (member 0 tablero)))
```

Jugadas del jugador "Computadora"

a) MOVIMIENTO-PC llama a ESCOGER-ESTRATEGIA-PC para seleccionar la estrategia a usar en las jugadas de la máquina; llama también a la función MOVIMIENTO-OPONENTE.

```
(defun movimiento-pc (tablero)
   (let* ((mejor-movimiento (escoger-estrategia-pc tablero))
          (pos (first mejor-movimiento))
          (estrategia (second mejor-movimiento))
          (nuevo-tablero (efectuar-movimiento *pc* pos tablero)))
     (format t "~&Mi movimiento: ~S" pos)
     (format t "~&Mi estrategia: ~A~%" estrategia)
     (imprimir-tablero nuevo-tablero)
     (cond ((ganador-p nuevo-tablero) (format t "~&Yo gano!"))
           ((tablero-total-p nuevo-tablero)(format t "~&Juego
empatado."))
           (t (movimiento-oponente nuevo-tablero)))
   )
)
```

b) Usaremos para la máquina una estrategia aleatoria: escoge al azar un movimiento legal mediante la función ESTRATEGIA-ALEATORIA, la cual, a su vez, usa SELECCIÓN-ALEATORIA-CASILLERO-VACIO para tomar un número aleatorio.

```
(defun escoger-estrategia-pc (tablero)
   (estrategia-aleatoria tablero))
```

```
(defun estrategia-aleatoria (tablero)
   (list (seleccion-aleatoria-casillero-vacio tablero) "movimiento
   aleatorio"))

(defun seleccion-aleatoria-casillero-vacio (tablero)
  (let ((pos (+ 1 (random 9))))
         (if (zerop (nth pos tablero)) pos (seleccion-aleatoria-
   casillero-vacio tablero))
  ))
```

Instrucciones para jugar

a) En el Lispworks copiar en un archivo nuevo el código incluido en el presente documento; luego guardarlo como `estrategia-aleatoria_michi_2016-1.lsp` en una carpeta.
b) Abrir el archivo generado en el ítem anterior según procedimiento rutinario:
b.1) En el Lispworks hacer clic en `File/Open…`;
b.2) Configurar el path del archivo fuente.
b.3) Hacer clic en botón `Open`.
c) Interpretar el código
d) Llamar a la función sin argumentos llamada `iniciar-juego` (aparece una ventana preguntando si uno desea iniciar el juego y dos botones: Yes y No).
e) Al hacer clic en Yes se muestra el tablero de juego y aparece mensaje Tu movimiento. Uno deberá indicar un número entre 1 al 9; no se indica que ellos denotan las posiciones del tablero.

Secuencia ejemplo
Enseguida se muestra la secuencia de posibles jugadas al aplicar la estrategia aleatoria:

CL-USER 5 > (Iniciar-juego)
Tu movimiento: 1

```
 O |   |
-----------
  |   |
-----------
  |   |
```

Mi movimiento: 5
Mi estrategia: movimiento aleatorio

```
 O |   |
-----------
  | X |
-----------
  |   |
```

Tu movimiento: 2

```
 O | O |
-----------
  | X |
-----------
  |   |
```

Mi movimiento: 3
Mi estrategia: movimiento aleatorio

```
 O | O | X
 ----------
   | X |
 ----------
   |   |
```

Tu movimiento: 4

```
 O | O | X
 ----------
 O | X |
 ----------
   |   |
```

Mi movimiento: 8
Mi estrategia: movimiento aleatorio

```
 O | O | X
 ----------
 O | X |
 ----------
   | X |
```

Tu movimiento: 7

```
 O | O | X
 ----------
 O | X |
 ----------
 O | X |
```

Ud. gano!
NIL

RAMP