Harry Scells
Craig Graci
30 January 2015

Ttt Lm: Visualisation, Analysis And Statistics

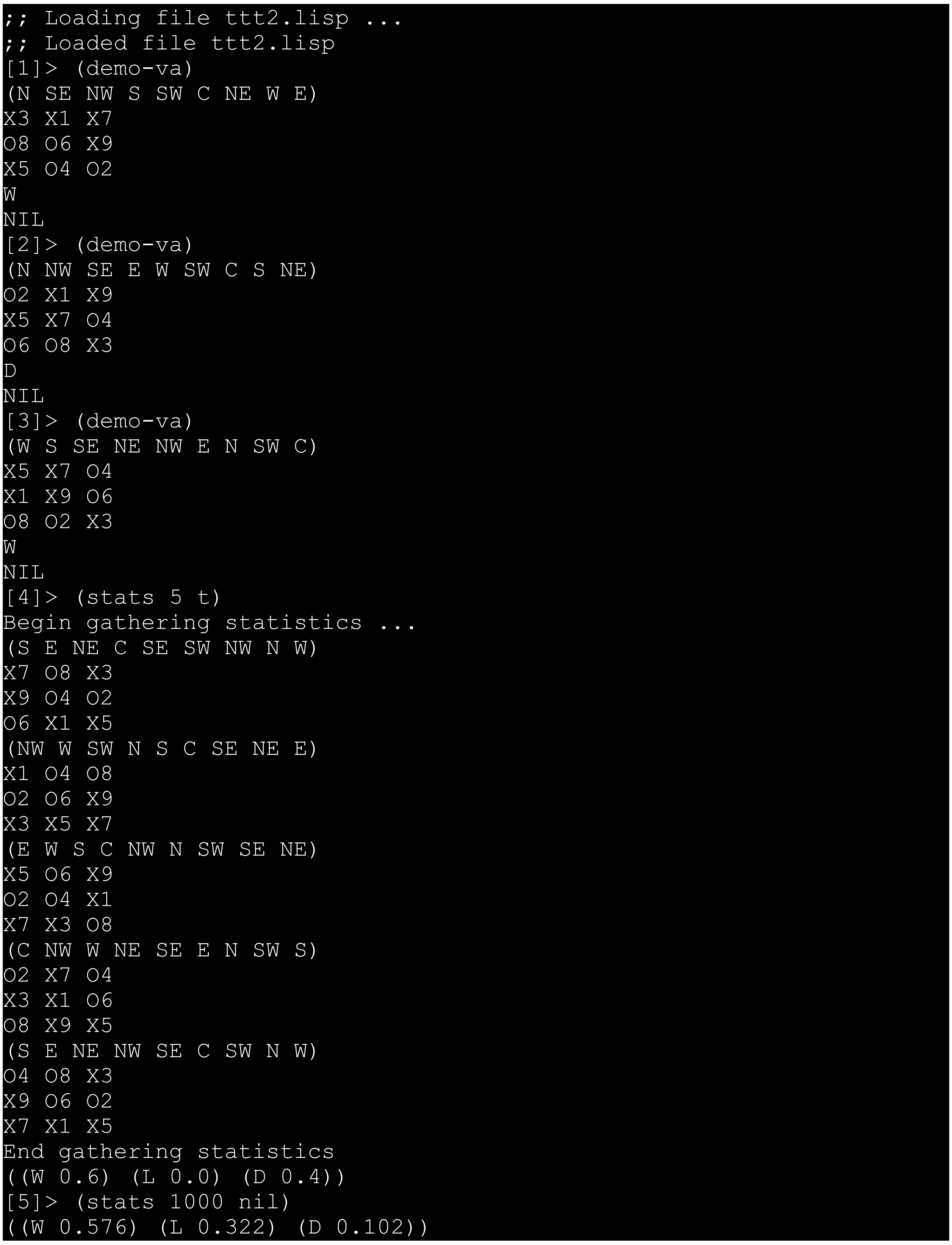
Code

This code outlines the two methods that were implemented.

```
; get the position of player in a game
(defun pos (obj lst)
  (loop for i in lst and position from 0 when (eql i obj) return position)
)

; get the player name from the play made
(defun playern (num)
   (cond ((eq 0 (mod num 2)) 'X)(t 'O))
)

; draws a single row of the board
(defun drawrow (moves lst)
   (dolist (move moves)
      (format t "~S~S " (playern(pos move lst)) (+ 1 (pos move lst)))
   )
)

(defun visualize (lst)
   (drawrow '(nw n ne) lst)
   (format t "~%")
   (drawrow '(w c e) lst)
   (format t "~%")
   (drawrow '(sw s se) lst)
   (format t "~%")
)

; sum the scores of a list
(defun sumscore (sublst lst)
   (setf sum 0)
   (dolist (l sublst)
      (setf sum (+ sum (pos l lst)))
   )
   (return-from sumscore sum)
)
```

```

; find the highest score for a given 'winning' outcome
(defun find-highest (play lst)
   (setf highest 0)
   (dolist (n play)
      (cond
         ((> (pos n lst) highest) (setf highest (pos n lst)))
      )
   )
   (return-from find-highest highest)
)

; analyse the final play
(defun analyze (lst)
   ; extract out each players moves
   (setf playo (list (nth 1 lst) (nth 3 lst) (nth 5 lst) (nth 7 lst)))
   (setf playx (list (nth 0 lst) (nth 2 lst) (nth 4 lst) (nth 6 lst) (nth 8 lst)))

   ; get a list of all the winning conditions
   (setf wcond '(
      (nw n ne) (e c w) (sw s se)
      (nw w sw) (n c s) (ne e se)
      (ne c sw) (nw c se)
   ))

   ; allow wins to be stored in a list
   (setf px '())
   (setf po '())
   ; move through the winning conditions and find the best play
   (dolist (w wcond)
      (cond
         ((eq (length (intersection w playx)) 3) (setf px w))
         ((eq (length (intersection w playo)) 3) (setf po w))
      )
   )
   (setf highestx (find-highest px lst))
   (setf highesto (find-highest po lst))
   ; determine the winner or if the game is a draw
   (cond
      ((and (eq px nil) (eq po nil)) (return-from analyze 'd))
      ((eq px nil) (return-from analyze 'l))
      ((eq po nil) (return-from analyze 'w))
      ((> highestx highesto) (return-from analyze 'l))
      ((< highestx highesto) (return-from analyze 'w))
   )
   (return-from analyze '?)
)
```

Demo



```
;; Loading file ttt2.lisp ...
;; Loaded file ttt2.lisp
[1]> (demo-va)
(N SE NW S SW C NE W E)
X3 X1 X7
O8 O6 X9
X5 O4 O2
W
NIL
[2]> (demo-va)
(N NW SE E W SW C S NE)
O2 X1 X9
X5 X7 O4
O6 O8 X3
D
NIL
[3]> (demo-va)
(W S SE NE NW E N SW C)
X5 X7 O4
X1 X9 O6
O8 O2 X3
W
NIL
[4]> (stats 5 t)
Begin gathering statistics ...
(S E NE C SE SW NW N W)
X7 O8 X3
X9 O4 O2
O6 X1 X5
(NW W SW N S C SE NE E)
X1 O4 O8
O2 O6 X9
X3 X5 X7
(E W S C NW N SW SE NE)
X5 O6 X9
O2 O4 X1
X7 X3 O8
(C NW W NE SE E N SW S)
O2 X7 O4
X3 X1 O6
O8 X9 X5
(S E NE NW SE C SW N W)
O4 O8 X3
X9 O6 O2
X7 X1 X5
End gathering statistics
((W 0.6) (L 0.0) (D 0.4))
[5]> (stats 1000 nil)
((W 0.576) (L 0.322) (D 0.102))
```

Results

It can be seen that by having player X move first, it gives that player a much higher advantage. In having five possible moves as opposed to four, there is a much higher chance of winning by pure chance, purely due to the fact that there are more possible spaces to enter. When run over a large data set it is clear to see that this is evident.