TTT LM: Heuristic Machine

Definition for a heuristic machine, and demo games.

Jacob Peck

CSC 466

Professor Craig Graci

Spring 2011

Listing of `ttt.l`:

```
;; tic-tac-toe machine learning

;; select
(defmethod select ((l list))
  (nth (random (length l)) l)
)

;; snoc
(defmethod snoc ((s symbol) (l list))
  (append l (list s))
)

;; play
(defmethod play (&aux play avail move)
  (setf play ())
  (setf avail '(nw n ne w c e sw s se))
  (dolist (player '(x o x o x o x o x))
    (cond
      ((eq player 'x)
        (setf move (select avail))
        (setf avail (remove move avail))
        (setf play (snoc move play))
      )
      ((eq player 'o)
        (setf move (select avail))
        (setf avail (remove move avail))
        (setf play (snoc move play))
      )
    )
  )
  play
)

;; helper class - board
(defclass board ()
  (
    (nw :accessor board-nw :initform "--")
    (n :accessor board-n :initform "--")
    (ne :accessor board-ne :initform "--")
    (w :accessor board-w :initform "--")
    (c :accessor board-c :initform "--")
    (e :accessor board-e :initform "--")
    (sw :accessor board-sw :initform "--")
    (s :accessor board-s :initform "--")
    (se :accessor board-se :initform "--")
  )
)

(defmethod populate-board ((l list) &aux board turnnum player)
  (setf board (make-instance 'board))
  (setf turnnum 1)
  (setf player 'x)
  (dolist (element l)
```

```
    ; go through the list, assigning move values to the board positions
    (cond
      ((eq element 'nw)
        (setf (board-nw board) (concatenate 'string (write-to-string player)
(write-to-string turnnum)))
      )
      ((eq element 'n)
        (setf (board-n board) (concatenate 'string (write-to-string player)
(write-to-string turnnum)))
      )
      ((eq element 'ne)
        (setf (board-ne board) (concatenate 'string (write-to-string player)
(write-to-string turnnum)))
      )
      ((eq element 'w)
        (setf (board-w board) (concatenate 'string (write-to-string player)
(write-to-string turnnum)))
      )
      ((eq element 'c)
        (setf (board-c board) (concatenate 'string (write-to-string player)
(write-to-string turnnum)))
      )
      ((eq element 'e)
        (setf (board-e board) (concatenate 'string (write-to-string player)
(write-to-string turnnum)))
      )
      ((eq element 'sw)
        (setf (board-sw board) (concatenate 'string (write-to-string player)
(write-to-string turnnum)))
      )
      ((eq element 's)
        (setf (board-s board) (concatenate 'string (write-to-string player)
(write-to-string turnnum)))
      )
      ((eq element 'se)
        (setf (board-se board) (concatenate 'string (write-to-string player)
(write-to-string turnnum)))
      )
    )
    (if (eq player 'x)
      (setf player 'o)
      ;else
      (progn (setf player 'x) (setf turnnum (+ 1 turnnum)))
    )
  )
  board
)

(defmethod analyze-board ((l list) &aux board value turnnum player)
  (setf board (make-instance 'board))
  (setf value 'd)
  (setf turnnum 1)
  (setf player 'x)
  (dolist (element l)
```

```
    ;; populate board, checking after each move for a win/loss/draw in terms
of player X
    (cond
      ((eq element 'nw)
        (setf (board-nw board) (concatenate 'string (write-to-string player)
(write-to-string turnnum)))
      )
      ((eq element 'n)
        (setf (board-n board) (concatenate 'string (write-to-string player)
(write-to-string turnnum)))
      )
      ((eq element 'ne)
        (setf (board-ne board) (concatenate 'string (write-to-string player)
(write-to-string turnnum)))
      )
      ((eq element 'w)
        (setf (board-w board) (concatenate 'string (write-to-string player)
(write-to-string turnnum)))
      )
      ((eq element 'c)
        (setf (board-c board) (concatenate 'string (write-to-string player)
(write-to-string turnnum)))
      )
      ((eq element 'e)
        (setf (board-e board) (concatenate 'string (write-to-string player)
(write-to-string turnnum)))
      )
      ((eq element 'sw)
        (setf (board-sw board) (concatenate 'string (write-to-string player)
(write-to-string turnnum)))
      )
      ((eq element 's)
        (setf (board-s board) (concatenate 'string (write-to-string player)
(write-to-string turnnum)))
      )
      ((eq element 'se)
        (setf (board-se board) (concatenate 'string (write-to-string player)
(write-to-string turnnum)))
      )
    )
    (if (eq player 'x)
      (setf player 'o)
      ;else
      (progn (setf player 'x) (setf turnnum (+ 1 turnnum)))
    )
    ; only change if game is currently a draw
    (if (eq value 'd) (setf value (check-for-win board)))
  )
  value
)

(defmethod check-for-win ((b board) &aux value)
  (setf value (check-for-win-row-1 b))
  (if (eq value 'd) (setf value (check-for-win-row-2 b)))
  (if (eq value 'd) (setf value (check-for-win-row-3 b)))
```

```
  (if (eq value 'd) (setf value (check-for-win-col-1 b)))
  (if (eq value 'd) (setf value (check-for-win-col-2 b)))
  (if (eq value 'd) (setf value (check-for-win-col-3 b)))
  (if (eq value 'd) (setf value (check-for-win-diag-1 b)))
  (if (eq value 'd) (setf value (check-for-win-diag-2 b)))
  value
)

(defmethod check-for-win-row-1 ((b board) &aux value xwin owin)
  (setf value 'd)
  (setf xwin
    (and
      (find #\x (board-nw b) :test #'equalp)
      (find #\x (board-n b) :test #'equalp)
      (find #\x (board-ne b) :test #'equalp)
    )
  )
  (setf owin
    (and
      (find #\o (board-nw b) :test #'equalp)
      (find #\o (board-n b) :test #'equalp)
      (find #\o (board-ne b) :test #'equalp)
    )
  )
  (if xwin (setf value 'w))
  (if owin (setf value 'l))
  value
)

(defmethod check-for-win-row-2 ((b board) &aux value xwin owin)
  (setf value 'd)
  (setf xwin
    (and
      (find #\x (board-w b) :test #'equalp)
      (find #\x (board-c b) :test #'equalp)
      (find #\x (board-e b) :test #'equalp)
    )
  )
  (setf owin
    (and
      (find #\o (board-w b) :test #'equalp)
      (find #\o (board-c b) :test #'equalp)
      (find #\o (board-e b) :test #'equalp)
    )
  )
  (if xwin (setf value 'w))
  (if owin (setf value 'l))
  value
)

(defmethod check-for-win-row-3 ((b board) &aux value xwin owin)
  (setf value 'd)
  (setf xwin
    (and
      (find #\x (board-sw b) :test #'equalp)
```

```
      (find #\x (board-s b) :test #'equalp)
      (find #\x (board-se b) :test #'equalp)
    )
  )
  (setf owin
    (and
      (find #\o (board-sw b) :test #'equalp)
      (find #\o (board-s b) :test #'equalp)
      (find #\o (board-se b) :test #'equalp)
    )
  )
  (if xwin (setf value 'w))
  (if owin (setf value 'l))
  value
)

(defmethod check-for-win-col-1 ((b board) &aux value xwin owin)
  (setf value 'd)
  (setf xwin
    (and
      (find #\x (board-nw b) :test #'equalp)
      (find #\x (board-w b) :test #'equalp)
      (find #\x (board-sw b) :test #'equalp)
    )
  )
  (setf owin
    (and
      (find #\o (board-nw b) :test #'equalp)
      (find #\o (board-w b) :test #'equalp)
      (find #\o (board-sw b) :test #'equalp)
    )
  )
  (if xwin (setf value 'w))
  (if owin (setf value 'l))
  value
)

(defmethod check-for-win-col-2 ((b board) &aux value xwin owin)
  (setf value 'd)
  (setf xwin
    (and
      (find #\x (board-n b) :test #'equalp)
      (find #\x (board-c b) :test #'equalp)
      (find #\x (board-s b) :test #'equalp)
    )
  )
  (setf owin
    (and
      (find #\o (board-n b) :test #'equalp)
      (find #\o (board-c b) :test #'equalp)
      (find #\o (board-s b) :test #'equalp)
    )
  )
  (if xwin (setf value 'w))
  (if owin (setf value 'l))
```

```
  value
)

(defmethod check-for-win-col-3 ((b board) &aux value xwin owin)
  (setf value 'd)
  (setf xwin
    (and
      (find #\x (board-ne b) :test #'equalp)
      (find #\x (board-e b) :test #'equalp)
      (find #\x (board-se b) :test #'equalp)
    )
  )
  (setf owin
    (and
      (find #\o (board-ne b) :test #'equalp)
      (find #\o (board-e b) :test #'equalp)
      (find #\o (board-se b) :test #'equalp)
    )
  )
  (if xwin (setf value 'w))
  (if owin (setf value 'l))
  value
)

(defmethod check-for-win-diag-1 ((b board) &aux value xwin owin)
  (setf value 'd)
  (setf xwin
    (and
      (find #\x (board-nw b) :test #'equalp)
      (find #\x (board-c b) :test #'equalp)
      (find #\x (board-se b) :test #'equalp)
    )
  )
  (setf owin
    (and
      (find #\o (board-nw b) :test #'equalp)
      (find #\o (board-c b) :test #'equalp)
      (find #\o (board-se b) :test #'equalp)
    )
  )
  (if xwin (setf value 'w))
  (if owin (setf value 'l))
  value
)

(defmethod check-for-win-diag-2 ((b board) &aux value xwin owin)
  (setf value 'd)
  (setf xwin
    (and
      (find #\x (board-ne b) :test #'equalp)
      (find #\x (board-c b) :test #'equalp)
      (find #\x (board-sw b) :test #'equalp)
    )
  )
  (setf owin
```

```
    (and
      (find #\o (board-ne b) :test #'equalp)
      (find #\o (board-c b) :test #'equalp)
      (find #\o (board-sw b) :test #'equalp)
    )
  )
  (if xwin (setf value 'w))
  (if owin (setf value 'l))
  value
)

(defmethod visualize ((b board))
  (format t "~A ~A ~A~%~A ~A ~A~%~A ~A ~A~%"
    (board-nw b) (board-n b) (board-ne b)
    (board-w b)  (board-c b) (board-e b)
    (board-sw b) (board-s b) (board-se b)
  )
  NIL
)

;; visualize
(defmethod visualize ((l list) &aux board)
  (setf board (populate-board l))
  (visualize board)
  NIL
)

;; analyze
(defmethod analyze ((l list))
  (analyze-board l)
)

;; demo
(defmethod demo (&aux p)
  (setf p (play))
  (format t "~A~%" p)
  (visualize p)
  (format t "~A~%" (analyze p))
  NIL
)

;; stats
(defmethod stats ((f function) (n number) (demo t) &aux w l d p result
results)
  (setf w 0 l 0 d 0)
  (dotimes (i n)
    (setf p (apply f ()))
    (if demo (format t "~A~%" p))
    (if demo (visualize p))
    (setf result (analyze p))
    (if demo (format t "~A~%" result))
    (cond
      ((eq result 'w) (setf w (+ 1 w)))
      ((eq result 'l) (setf l (+ 1 l)))
      ((eq result 'd) (setf d (+ 1 d)))
```

```
    )
  )
  (setf results (mapcar #'probability (list w l d) (list n n n)))
  (mapcar #'list '(w l d) results)
)

;; probability
(defmethod probability ((special integer) (total integer))
  (/ (float special) (float total))
)

;; player classes
(defclass player ()
  (
    (name :accessor player-name :initarg :name)
  )
)

(defclass random-machine-player (player) ())

(defclass human-player (player) ())

(defclass heuristic-machine-player (player)
  (
    (rules :accessor heuristic-machine-player-rules :initarg :rules :initform
'())
  )
)


;; generic play
(defmethod generic-play ((x player) (o player) &aux move)
  (setf *avail* '(nw n ne w c e sw s se))
  (setf *play-so-far* ())
  (dolist (player '(x o x o x o x o x))
    (cond
      ((eq player 'x)
        (setf move (make-move x))
      )
      ((eq player 'o)
        (setf move (make-move o))
      )
    )
    (setf *play-so-far* (snoc move *play-so-far*))
    (if (game-over-p *play-so-far*) (return-from generic-play *play-so-far*))
  )
  *play-so-far*
)

;; rmp make move
(defmethod make-move ((p random-machine-player) &aux move)
  (setf move (select *avail*))
  (setf *avail* (remove move *avail*))
  move
)
```

```
;; rmp-rmp game
(defmethod demo-random-random (&aux p x o)
  (setf x (make-instance 'random-machine-player))
  (setf o (make-instance 'random-machine-player))
  (setf p (generic-play x o))
  (format t "~A~%" p)
  (visualize p)
  (format t "~A~%" (analyze p))
  NIL
)

;; human make move
(defmethod make-move ((p human-player) &aux move)
  (format t "BEGIN HUMAN PLAYER MOVE ...~%")
  (format t "Play so far = ~A~%" *play-so-far*)
  (visualize *play-so-far*)
  (format t "Please select a move from ~A~%" *avail*)
  (setf move (read))
  (cond
    ((not (member move *avail*))
      (make-move p)
    )
    (t
      (setf *avail* (remove move *avail*))
      move
    )
  )
  (format t "END HUMAN PLAYER MOVE~%")
  move
)

;; rmp-human game
(defmethod demo-random-human (&aux p x o)
  (setf x (make-instance 'random-machine-player))
  (setf o (make-instance 'human-player))
  (setf p (generic-play x o))
  (format t "~A~%" p)
  (visualize p)
  (format t "~A~%" (analyze p))
  NIL
)

;; heuristic machine player move
(defmethod make-move ((p heuristic-machine-player) &aux rule move)
  (format t "BEGIN HEURISTIC PLAYER MOVE ...~%")
  (setf rule (select-from-rule-base p))
  (if (null rule)
    (let ()
      (setf move (select *avail*))
      (format t "Making random move ~A since no rule is applicable.~%" move)
    )
    (let ()
      (setf move (apply-rule rule))
      (format t "Play so far = ~A~%" *play-so-far*)
```

```
      (format t "Making move ~A by applying the rule: ~A~%" move rule)
    )
  )
  (setf *avail* (remove move *avail*))
  (format t "END HEURISTIC PLAYER MOVE~%")
  move
)

;; heuristic select move
(defmethod select-from-rule-base ((p heuristic-machine-player) &aux rule-
base)
  (setf rule-base (heuristic-machine-player-rules p))
  (dolist (rule rule-base)
    (cond
      ((applicablep rule)
        (return-from select-from-rule-base rule)
      )
    )
  )
  NIL
)

;; applicablep
(defmethod applicablep ((rule list) &aux the-play)
  (setf the-play (third (second rule)))
  (matches *play-so-far* the-play)
)

;; apply-rule
(defmethod apply-rule ((rule list) &aux the-play)
  (setf the-play (fourth (fourth rule)))
  (nth (length *play-so-far*) the-play)
)

;; hmp-human game
(defmethod demo-heuristic-human ((nr-rules integer) &aux p x o)
  (setf x (make-instance 'heuristic-machine-player :name 'hm))
  (add-rules x nr-rules)
  (display x)
  (setf o (make-instance 'human-player :name 'hu))
  (display o)
  (setf p (generic-play x o))
  (format t "GAME SUMMARY~%")
  (format t "Play of the game = ~A~%" p)
  (visualize p)
  (format t "~A~%" (analyze p))
  NIL
)

;; hmp make rule
(defmethod make-rule ((l list))
  (list 'if (list 'prefix 'of l 'matches 'the 'play 'so 'far)
    'then (list 'select 'move 'from l))
)
```

```
;; matches
(defmethod matches ((l1 list) (l2 list))
  (cond
    ((null l1)
      t
    )
    ((eq (car l1) (car l2))
      (matches (cdr l1) (cdr l2))
    )
    (t
      NIL
    )
  )
)

;; human player display
(defmethod display ((p human-player))
  (format t "HUMAN PLAYER ...~%")
  (format t "name = ~A~%" (player-name p))
  NIL
)

;; hmp display
(defmethod display ((p heuristic-machine-player) &aux rules)
  (setf rules (heuristic-machine-player-rules p))
  (format t "HEURISTIC MACHINE PLAYER ...~%")
  (format t "name = ~A~%" (player-name p))
  (format t "rules ...~%")
  (dolist (rule rules)
    (format t "~A~%" rule)
  )
  NIL
)

;; add-rules
(defmethod add-rules ((p heuristic-machine-player) (num integer) &aux temp
rules)
  (setf rules (heuristic-machine-player-rules p))
  (loop while (< (length rules) num) do
    (setf temp (play))
    (if (eq 'W (analyze temp))
        (push (make-rule temp) rules)
    )
  )
  (setf (heuristic-machine-player-rules p) rules)
  NIL
)

;; game-over-p
(defmethod game-over-p ((l list))
  (if (eq (analyze l) 'D) NIL t)
)
```

Listing of `ttt-rmp-human-demo.text`:

```
$ clisp

<...snip...>

[1]> (load "ttt.l")
;; Loading file ttt.l ...
;; Loaded file ttt.l
T
[2]> (demo-heuristic-human 25)
HEURISTIC MACHINE PLAYER ...
name = HM
rules ...
(IF (PREFIX OF (E NE NW S SE N C SW W) MATCHES THE PLAY SO FAR) THEN
 (SELECT MOVE FROM (E NE NW S SE N C SW W)))
(IF (PREFIX OF (NE SW W SE S N E NW C) MATCHES THE PLAY SO FAR) THEN
 (SELECT MOVE FROM (NE SW W SE S N E NW C)))
(IF (PREFIX OF (W NW C S SW N SE E NE) MATCHES THE PLAY SO FAR) THEN
 (SELECT MOVE FROM (W NW C S SW N SE E NE)))
(IF (PREFIX OF (W S NW C N SW NE SE E) MATCHES THE PLAY SO FAR) THEN
 (SELECT MOVE FROM (W S NW C N SW NE SE E)))
(IF (PREFIX OF (C SE E N S SW W NE NW) MATCHES THE PLAY SO FAR) THEN
 (SELECT MOVE FROM (C SE E N S SW W NE NW)))
(IF (PREFIX OF (NW S C N E NE W SW SE) MATCHES THE PLAY SO FAR) THEN
 (SELECT MOVE FROM (NW S C N E NE W SW SE)))
(IF (PREFIX OF (C NE S W NW SW E N SE) MATCHES THE PLAY SO FAR) THEN
 (SELECT MOVE FROM (C NE S W NW SW E N SE)))
(IF (PREFIX OF (SE S NE N C E SW W NW) MATCHES THE PLAY SO FAR) THEN
 (SELECT MOVE FROM (SE S NE N C E SW W NW)))
(IF (PREFIX OF (S C NW NE SE E W N SW) MATCHES THE PLAY SO FAR) THEN
 (SELECT MOVE FROM (S C NW NE SE E W N SW)))
(IF (PREFIX OF (W NE NW S C N SW E SE) MATCHES THE PLAY SO FAR) THEN
 (SELECT MOVE FROM (W NE NW S C N SW E SE)))
(IF (PREFIX OF (N E SE S C W SW NW NE) MATCHES THE PLAY SO FAR) THEN
 (SELECT MOVE FROM (N E SE S C W SW NW NE)))
(IF (PREFIX OF (NW SW S W N NE SE E C) MATCHES THE PLAY SO FAR) THEN
 (SELECT MOVE FROM (NW SW S W N NE SE E C)))
(IF (PREFIX OF (E NW C SW W N NE S SE) MATCHES THE PLAY SO FAR) THEN
 (SELECT MOVE FROM (E NW C SW W N NE S SE)))
(IF (PREFIX OF (N S E C W NE SW SE NW) MATCHES THE PLAY SO FAR) THEN
 (SELECT MOVE FROM (N S E C W NE SW SE NW)))
(IF (PREFIX OF (N S SW W E SE NW C NE) MATCHES THE PLAY SO FAR) THEN
 (SELECT MOVE FROM (N S SW W E SE NW C NE)))
(IF (PREFIX OF (SW N SE NW C W NE E S) MATCHES THE PLAY SO FAR) THEN
 (SELECT MOVE FROM (SW N SE NW C W NE E S)))
(IF (PREFIX OF (SW N NE E W C SE NW S) MATCHES THE PLAY SO FAR) THEN
 (SELECT MOVE FROM (SW N NE E W C SE NW S)))
(IF (PREFIX OF (N NE S W NW SE E SW C) MATCHES THE PLAY SO FAR) THEN
 (SELECT MOVE FROM (N NE S W NW SE E SW C)))
(IF (PREFIX OF (SE N C E NW SW W S NE) MATCHES THE PLAY SO FAR) THEN
 (SELECT MOVE FROM (SE N C E NW SW W S NE)))
(IF (PREFIX OF (S NE NW N E C SW SE W) MATCHES THE PLAY SO FAR) THEN
 (SELECT MOVE FROM (S NE NW N E C SW SE W)))
```

```
(IF (PREFIX OF (SW S NE W N C NW E SE) MATCHES THE PLAY SO FAR) THEN
 (SELECT MOVE FROM (SW S NE W N C NW E SE)))
(IF (PREFIX OF (S W E NE SW NW SE C N) MATCHES THE PLAY SO FAR) THEN
 (SELECT MOVE FROM (S W E NE SW NW SE C N)))
(IF (PREFIX OF (NE NW SW N SE C W E S) MATCHES THE PLAY SO FAR) THEN
 (SELECT MOVE FROM (NE NW SW N SE C W E S)))
(IF (PREFIX OF (NW E W N NE S SE SW C) MATCHES THE PLAY SO FAR) THEN
 (SELECT MOVE FROM (NW E W N NE S SE SW C)))
(IF (PREFIX OF (SW NE N NW S C SE E W) MATCHES THE PLAY SO FAR) THEN
 (SELECT MOVE FROM (SW NE N NW S C SE E W)))
HUMAN PLAYER ...
name = HU
BEGIN HEURISTIC PLAYER MOVE ...
Play so far = NIL
Making move E by applying the rule:
(IF (PREFIX OF (E NE NW S SE N C SW W) MATCHES THE PLAY SO FAR) THEN
 (SELECT MOVE FROM (E NE NW S SE N C SW W)))
END HEURISTIC PLAYER MOVE
BEGIN HUMAN PLAYER MOVE ...
Play so far = (E)
-- -- --
-- -- X1
-- -- --
Please select a move from (NW N NE W C SW S SE)
NE
END HUMAN PLAYER MOVE
BEGIN HEURISTIC PLAYER MOVE ...
Play so far = (E NE)
Making move NW by applying the rule:
(IF (PREFIX OF (E NE NW S SE N C SW W) MATCHES THE PLAY SO FAR) THEN
 (SELECT MOVE FROM (E NE NW S SE N C SW W)))
END HEURISTIC PLAYER MOVE
BEGIN HUMAN PLAYER MOVE ...
Play so far = (E NE NW)
X2 -- O1
-- -- X1
-- -- --
Please select a move from (N W C SW S SE)
S
END HUMAN PLAYER MOVE
BEGIN HEURISTIC PLAYER MOVE ...
Play so far = (E NE NW S)
Making move SE by applying the rule:
(IF (PREFIX OF (E NE NW S SE N C SW W) MATCHES THE PLAY SO FAR) THEN
 (SELECT MOVE FROM (E NE NW S SE N C SW W)))
END HEURISTIC PLAYER MOVE
BEGIN HUMAN PLAYER MOVE ...
Play so far = (E NE NW S SE)
X2 -- O1
-- -- X1
-- O2 X3
Please select a move from (N W C SW)
N
END HUMAN PLAYER MOVE
BEGIN HEURISTIC PLAYER MOVE ...
```

```
Play so far = (E NE NW S SE N)
Making move C by applying the rule:
(IF (PREFIX OF (E NE NW S SE N C SW W) MATCHES THE PLAY SO FAR) THEN
 (SELECT MOVE FROM (E NE NW S SE N C SW W)))
END HEURISTIC PLAYER MOVE
GAME SUMMARY
Play of the game = (E NE NW S SE N C)
X2 O3 O1
-- X4 X1
-- O2 X3
W
NIL
[3]> (demo-heuristic-human 25)
HEURISTIC MACHINE PLAYER ...
name = HM
rules ...
(IF (PREFIX OF (NE N C SW W E NW S SE) MATCHES THE PLAY SO FAR) THEN
 (SELECT MOVE FROM (NE N C SW W E NW S SE)))
(IF (PREFIX OF (NE S W SE NW C N E SW) MATCHES THE PLAY SO FAR) THEN
 (SELECT MOVE FROM (NE S W SE NW C N E SW)))
(IF (PREFIX OF (C N W S E NW SE SW NE) MATCHES THE PLAY SO FAR) THEN
 (SELECT MOVE FROM (C N W S E NW SE SW NE)))
(IF (PREFIX OF (E N C NE W SE S SW NW) MATCHES THE PLAY SO FAR) THEN
 (SELECT MOVE FROM (E N C NE W SE S SW NW)))
(IF (PREFIX OF (NW S NE C N SW SE W E) MATCHES THE PLAY SO FAR) THEN
 (SELECT MOVE FROM (NW S NE C N SW SE W E)))
(IF (PREFIX OF (NE W N C SW SE NW S E) MATCHES THE PLAY SO FAR) THEN
 (SELECT MOVE FROM (NE W N C SW SE NW S E)))
(IF (PREFIX OF (E SW C NW N S W SE NE) MATCHES THE PLAY SO FAR) THEN
 (SELECT MOVE FROM (E SW C NW N S W SE NE)))
(IF (PREFIX OF (C W N SE SW E S NW NE) MATCHES THE PLAY SO FAR) THEN
 (SELECT MOVE FROM (C W N SE SW E S NW NE)))
(IF (PREFIX OF (NE SE C E NW W SW S N) MATCHES THE PLAY SO FAR) THEN
 (SELECT MOVE FROM (NE SE C E NW W SW S N)))
(IF (PREFIX OF (N NE E NW SW SE W S C) MATCHES THE PLAY SO FAR) THEN
 (SELECT MOVE FROM (N NE E NW SW SE W S C)))
(IF (PREFIX OF (SE SW NW W C N E S NE) MATCHES THE PLAY SO FAR) THEN
 (SELECT MOVE FROM (SE SW NW W C N E S NE)))
(IF (PREFIX OF (C SW SE E NW N S NE W) MATCHES THE PLAY SO FAR) THEN
 (SELECT MOVE FROM (C SW SE E NW N S NE W)))
(IF (PREFIX OF (SE E C W SW NE NW N S) MATCHES THE PLAY SO FAR) THEN
 (SELECT MOVE FROM (SE E C W SW NE NW N S)))
(IF (PREFIX OF (NE W C SW SE S E NW N) MATCHES THE PLAY SO FAR) THEN
 (SELECT MOVE FROM (NE W C SW SE S E NW N)))
(IF (PREFIX OF (SE S NW SW C NE W N E) MATCHES THE PLAY SO FAR) THEN
 (SELECT MOVE FROM (SE S NW SW C NE W N E)))
(IF (PREFIX OF (S E NE SE SW N NW C W) MATCHES THE PLAY SO FAR) THEN
 (SELECT MOVE FROM (S E NE SE SW N NW C W)))
(IF (PREFIX OF (E SE SW N W NW C S NE) MATCHES THE PLAY SO FAR) THEN
 (SELECT MOVE FROM (E SE SW N W NW C S NE)))
(IF (PREFIX OF (C N NE E SE NW SW S W) MATCHES THE PLAY SO FAR) THEN
 (SELECT MOVE FROM (C N NE E SE NW SW S W)))
(IF (PREFIX OF (NW C NE E N W SE SW S) MATCHES THE PLAY SO FAR) THEN
 (SELECT MOVE FROM (NW C NE E N W SE SW S)))
(IF (PREFIX OF (C N SE SW S W NW E NE) MATCHES THE PLAY SO FAR) THEN
```

```
 (SELECT MOVE FROM (C N SE SW S W NW E NE)))
(IF (PREFIX OF (N S NW E W NE SW C SE) MATCHES THE PLAY SO FAR) THEN
 (SELECT MOVE FROM (N S NW E W NE SW C SE)))
(IF (PREFIX OF (N S NW C NE E SE W SW) MATCHES THE PLAY SO FAR) THEN
 (SELECT MOVE FROM (N S NW C NE E SE W SW)))
(IF (PREFIX OF (SW S NE SE C W NW E N) MATCHES THE PLAY SO FAR) THEN
 (SELECT MOVE FROM (SW S NE SE C W NW E N)))
(IF (PREFIX OF (NE NW W SW E SE C S N) MATCHES THE PLAY SO FAR) THEN
 (SELECT MOVE FROM (NE NW W SW E SE C S N)))
(IF (PREFIX OF (N E NE S SE W SW NW C) MATCHES THE PLAY SO FAR) THEN
 (SELECT MOVE FROM (N E NE S SE W SW NW C)))
HUMAN PLAYER ...
name = HU
BEGIN HEURISTIC PLAYER MOVE ...
Play so far = NIL
Making move NE by applying the rule:
(IF (PREFIX OF (NE N C SW W E NW S SE) MATCHES THE PLAY SO FAR) THEN
 (SELECT MOVE FROM (NE N C SW W E NW S SE)))
END HEURISTIC PLAYER MOVE
BEGIN HUMAN PLAYER MOVE ...
Play so far = (NE)
-- -- X1
-- -- --
-- -- --
Please select a move from (NW N W C E SW S SE)
NW
END HUMAN PLAYER MOVE
BEGIN HEURISTIC PLAYER MOVE ...
Play so far = (NE NW)
Making move W by applying the rule:
(IF (PREFIX OF (NE NW W SW E SE C S N) MATCHES THE PLAY SO FAR) THEN
 (SELECT MOVE FROM (NE NW W SW E SE C S N)))
END HEURISTIC PLAYER MOVE
BEGIN HUMAN PLAYER MOVE ...
Play so far = (NE NW W)
O1 -- X1
X2 -- --
-- -- --
Please select a move from (N C E SW S SE)
C
END HUMAN PLAYER MOVE
BEGIN HEURISTIC PLAYER MOVE ...
Making random move S since no rule is applicable.
END HEURISTIC PLAYER MOVE
BEGIN HUMAN PLAYER MOVE ...
Play so far = (NE NW W C S)
O1 -- X1
X2 O2 --
-- X3 --
Please select a move from (N E SW SE)
SE
END HUMAN PLAYER MOVE
GAME SUMMARY
Play of the game = (NE NW W C S SE)
O1 -- X1
```

```
X2 O2 --
-- X3 O3
L
NIL
[4]> (bye)
Bye.
```