```
;;; 三目並べゲーム
;;;  originally written by D. Touretzky

;;; 三並び配置の三つ組 (大域変数)
;;;   1 | 2 | 3
;;;  -----------
;;;   4 | 5 | 6
;;;  -----------
;;;   7 | 8 | 9
(setf *triplets*
  '((1 2 3) (4 5 6) (7 8 9)   ; 水平
    (1 4 7) (2 5 8) (3 6 9)   ; 垂直
    (1 5 9) (3 5 7)))         ; 対角線


;;; 初期状態の作成
;;; 石が置かれていないのですべてのマスが0
;;; nth関数が0オリジンなのでダミーとして最初に'boardを入れる
(defun make-board ()
  (list 'board 0 0 0 0 0 0 0 0 0))


;;; 大域変数の初期化
(setf *human* 1)
(setf *computer* 10)


;;; 8通りすべてについて並びの状態を計算する
(defun compute-sums (board)
  (mapcar #'(lambda (triplet)
              (sum-triplet board triplet))
          *triplets*))


;;; 3並びの状態を計算する
(defun sum-triplet (board triplet)
  (+ (nth (first triplet) board)
     (nth (second triplet) board)
     (nth (third triplet) board)))


;;; 盤面を表示する
(defun print-board (board)
  (format t "~%")
  (print-row (nth 1 board) (nth 2 board) (nth 3 board))
  (format t "~&  -----------")
  (print-row (nth 4 board) (nth 5 board) (nth 6 board))
  (format t "~&  -----------")
  (print-row (nth 7 board) (nth 8 board) (nth 9 board))
  (format t "~%~%"))


;;; 1行分の状態を表示する
(defun print-row (x y z)
  (format t "~&   ~A | ~A | ~A"
       (convert-to-letter x)
       (convert-to-letter y)
       (convert-to-letter z)))


;;; 内部表現から画面表示への変換
;;; Oが人間, Xがコンピュータ
(defun convert-to-letter (v)
  (cond ((equal v *human*) "O")
        ((equal v *computer*) "X")
        (t " ")))
```

```
;;; 主関数
(defun playgame ()
  (format t "~&   1 | 2 | 3")
  (format t "~&  -----------")
  (format t "~&   4 | 5 | 6")
  (format t "~&  -----------")
  (format t "~&   7 | 8 | 9")
  (if (y-or-n-p "先手をとりますか?")
      (human-move (make-board))
      (computer-move (make-board))))


;;; 人間の手番
(defun human-move (board)
  (let* ((pos (read-a-legal-move board))
         (new-board (make-move *human* pos board)))
     (print-board new-board)
     (cond ((winner-p new-board)
            (format t "~&あなたの勝ち!"))
           ((board-full-p new-board)
            (format t "~&引き分け"))
           (t (computer-move new-board)))))


;;; 人間の次の手を入力させる
;;; 返り値は石の位置
(defun read-a-legal-move (board)
  (format t "~&あなたの手は? (1-9): ")
  (let ((pos (read)))
    (cond ((not (and (integerp pos)
                     (<= 1 pos)
                     (<= pos 9)))
           (format t "~&もう一度入力してください")
           (read-a-legal-move board))
          ((not (zerop (nth pos board)))
           (format t "~&そこはすでに置かれています")
           (read-a-legal-move board))
          (t pos))))


;;; 石を置く
;;; 返り値は新たな盤の状態
(defun make-move (player pos board)
  (setf (nth pos board) player)   ; 副作用が心配?
  board)


;;; コンピュータの手番
(defun computer-move (board)
  (let* ((best-move (choose-best-move board))
         (pos (first best-move))
         (strategy (second best-move))
         (new-board (make-move *computer* pos board)))
    (format t "~&こちらの手は ~S" pos)
    (format t "~&(戦略は~A)" strategy)
    (print-board new-board)
    (cond ((winner-p new-board)
           (format t "~&わたしの勝ち!"))
          ((board-full-p new-board)
           (format t "引き分け"))
          (t (human-move new-board)))))


;;; どちらかが勝ったかどうかの判定
(defun winner-p (board)
  (let ((sums (compute-sums board)))
     (or (member (* 3 *computer*) sums)
         (member (* 3 *human*) sums))))
```

```
;;; 盤面が全部埋まっているかどうかの判定
(defun board-full-p (board)
  (not (member 0 board)))


;;; コンピュータの次の手を選ぶ
;;; 戦略が大事!!
(defun choose-best-move (board)
  (or (make-three-in-a-row board)    ; 第一の戦略
      (block-human-win board)        ; 第二の戦略
      (random-move board)))          ; 最後の手段


;;; 一列に並ぶような手を打つ戦略
;;; 返り値は石の位置と戦略
(defun make-three-in-a-row (board)
  (let ((pos (win-or-block board
                           (* 2 *computer*))))
    (and pos (list pos "一列並べる"))))


;;; 人間の勝ちを妨害する手を打つ戦略
;;; 返り値は石の位置と戦略
(defun block-human-win (board)
  (let ((pos (win-or-block board
                           (* 2 *human*))))
    (and pos (list pos "勝ちを妨害"))))


;;; あと1つで三並びが達成される状態ならばその位置を返す
(defun win-or-block (board target-sum)
  (let ((triplet (find-if #'(lambda (trip)
                              (equal (sum-triplet board trip)
                                     target-sum))
                          *triplets*)))
       (if triplet (find-empty-position board triplet))))


;;; 並びの中の空いている位置を返す
(defun find-empty-position (board squares)
  (find-if #'(lambda (pos) (zerop (nth pos board)))
           squares))


;;; ランダムに手を打つ戦略
(defun random-move (board)
  (list (pick-random-position board)
        "ランダム"))


;;; 空所にランダムに手を選ぶ
(defun pick-random-position (board)
  (let ((pos (+ 1 (random 9))))
    (if (zerop (nth pos board))
        pos
        (pick-random-position board))))



; 上のプログラムは
;      wget http://www.nak.ics.keio.ac.jp/class/lisp/3moku.lsp
; で取得できます.
```