Conway's game of Life

```
--  The program in this chapter plays Life on a terminal screen.
--

--
--  configuration constants
--

VAL INT array.width  IS ... :   -- number of cells across the board
VAL INT array.height IS ... :   -- number of cells down the board

VAL INT radius     IS 1 :       -- of the 'sphere of influence'
VAL INT diameter   IS (2 * radius) + 1 :
VAL INT neighbours IS (diameter * diameter) - 1 :

VAL INT number.of.cells IS array.height * array.width :
VAL INT number.of.links IS neighbours * number.of.cells :

--
--  protocols
--

PROTOCOL STATE IS BOOL :

VAL BOOL alive IS TRUE :
VAL BOOL dead  IS NOT alive :

PROTOCOL COMMAND
  CASE
    set.state; BOOL
    evolve
    terminate
:

PROTOCOL RESPONSE IS BOOL; BOOL :
```

```
--
--  cell processes
--

PROC broadcast.present.state([][][]CHAN OF STATE link,
                             VAL INT x, y, VAL BOOL state )
  PAR d = 0 FOR neighbours
    link[x][y][d] ! state
:

PROC calculate.next.state([][][]CHAN OF STATE link,
                          VAL []INT nx, ny,
                          VAL BOOL state, BOOL next.state )
  INT count :       -- number of living neighbours
  SEQ
    [neighbours]BOOL state.of.neighbour :
    SEQ
      PAR d = 0 FOR neighbours
        link[nx[d]][ny[d]][d] ? state.of.neighbour[d]
      count := 0
      SEQ d = 0 FOR neighbours
        IF
          state.of.neighbour[d] = alive
            count := count + 1
          state.of.neighbour[d] = dead
            SKIP
    IF
      count < 2     -- death from isolation
        next.state := dead
      count = 2     -- this cell is stable
        next.state := state
      count = 3     -- stable if alive, a birth if dead
        next.state := alive
      count > 3     -- death from overcrowding
        next.state := dead
:
```

```
PROC cell([][][]CHAN OF STATE link,
          VAL INT x, y, VAL []INT nx, ny,
          CHAN OF COMMAND control,
          CHAN OF RESPONSE sense        )
  BOOL state, not.finished :
  SEQ
    state := dead       -- the whole board starts off dead
    not.finished := TRUE
    WHILE not.finished
      control ? CASE

        set.state; state
          SKIP          -- state has been set to the new value

        evolve
          BOOL next.state :
          SEQ
            PAR
              broadcast.present.state(link, x, y, state)
              SEQ
                calculate.next.state(link, nx, ny,
                                     state, next.state )
                sense ! (state <> next.state); next.state
            state := next.state

        terminate
          not.finished := FALSE
:

--
--  terminal-dependent output routines
--

PROC clear.screen(CHAN OF BYTE terminal)
  -- clear screen sequence for an ANSI terminal
  write.string(terminal, "*#1B[2J")
:

PROC move.cursor(CHAN OF BYTE terminal, VAL INT x, y)
  -- left-handed co-ordinates, origin 0,0 at top left
  CHAN OF DATA.ITEM c :
  PAR
    write.formatted(terminal, "*#1B[%d;%dH", c)
    SEQ
      c ! data.int; y + 1
      c ! data.int; x + 1
:
```

```
--
--  display routines
--

PROC initialize.display(CHAN OF BYTE screen)
  -- display an entirely dead board
  clear.screen(screen)
:

PROC clean.up.display(CHAN OF BYTE screen)
  move.cursor(screen, 0, array.height)
:

PROC display.state(CHAN OF BYTE screen, VAL INT x, y, VAL BOOL state)
  SEQ
    move.cursor(screen, x, y)
    IF
      state = alive
        screen ! '**'
      state = dead
        screen ! '*s'
:

--
--  controller states
--

VAL INT idle         IS 0 :  -- controller activity values
VAL INT editing      IS 1 :
VAL INT single.step  IS 2 :
VAL INT free.running IS 3 :
VAL INT terminated   IS 4 :


INT FUNCTION new.activity(VAL BYTE char)
  INT activity :
  VALOF
    CASE char     -- typed on the keyboard ...
      'q', 'Q'              -- ... Q to finish program
        activity := terminated
      's', 'S'              -- ... S to halt evolution
        activity := idle
      'e', 'E'              -- ... E to start editing
        activity := editing
      'r', 'R'              -- ... R to start evolution
        activity := free.running
      ELSE  -- ... or anything else for one generation
        activity := single.step
    RESULT activity
:
```

```
PROC display.activity(CHAN OF BYTE screen, VAL INT activity)
  SEQ
    move.cursor(screen, array.width+1, array.height/2)
    CASE activity
      idle
        write.string(screen, "Idle")
      editing
        write.string(screen, "Edit")
      single.step
        write.string(screen, "Step")
      free.running
        write.string(screen, "Busy")
      terminated
        write.string(screen, "Done")
:

--
--  generation
--

PROC generation(CHAN OF BYTE screen,
                [][]CHAN OF COMMAND control,
                [][]CHAN OF RESPONSE sense,
                BOOL active                 )
  SEQ
    PAR x = 0 FOR array.width
      PAR y = 0 FOR array.height
        control[x][y] ! evolve
    active := FALSE
    SEQ x = 0 FOR array.width
      SEQ y = 0 FOR array.height
        BOOL changed, next.state :
        SEQ
          sense[x][y] ? changed; next.state
          IF
            changed
              SEQ
                display.state(screen, x, y, next.state)
                active := TRUE
            NOT changed
              SKIP
:
```

```
--
--  editor
--

INT FUNCTION min(VAL INT a, b)
  INT min :
  VALOF
    IF
      a <= b
        min := a
      b <= a
        min := b
    RESULT min
:

INT FUNCTION max(VAL INT a, b)
  INT max :
  VALOF
    IF
      a >= b
        max := a
      b >= a
        max := b
    RESULT max
:
```

```
PROC editor(CHAN OF BYTE keyboard, screen,
            [][]CHAN OF COMMAND control   )
  INT x, y :
  BOOL editing :
  SEQ
    -- initialize co-ordinates to centre of board
    x, y := array.width / 2, array.height / 2
    editing := TRUE
    WHILE editing
      BYTE char :
      SEQ
        move.cursor(screen, x, y)
        keyboard ? char
        CASE char
          'A'        -- move up, if possible
            y := max(y - 1, 0)
          'B'        -- move down, if possible
            y := min(y + 1, array.height - 1)
          'C'        -- move right, if possible
            x := min(x + 1, array.width - 1)
          'D'        -- move left, if possible
            x := max(x - 1, 0)
          '*s', '**'
            VAL BOOL state IS (char = '**') = alive :
            PAR
              control[x][y] ! set.state; state
              display.state(screen, x, y, state)
          'q', 'Q'
            editing := FALSE
          ELSE
            SKIP     -- ignore anything else
:
```

```
--
--  controller
--

PROC controller(CHAN OF BYTE keyboard, screen,
                [][]CHAN OF COMMAND control,
                [][]CHAN OF RESPONSE sense    )
  INT activity :
  SEQ
    activity := idle
    initialize.display(screen)
    WHILE activity <> terminated
      SEQ
        display.activity(screen, activity)
        BYTE char :
        PRI ALT
          (activity <> editing) & keyboard ? char
            activity := new.activity(char)
          (activity <> idle) & SKIP
            CASE activity
              editing
                SEQ
                  editor(keyboard, screen, control)
                  activity := idle
              free.running, single.step
                BOOL changing :
                SEQ
                  generation(screen, control, sense, changing)
                  IF
                    (activity = single.step) OR (NOT changing)
                      activity := idle
                    (activity = free.running) AND changing
                      SKIP
    display.activity(screen, activity)
    PAR x = 0 FOR array.width
      PAR y = 0 FOR array.height
        control[x][y] ! terminate
    clean.up.display(screen)
:
```

```
--
--  structure of the program
--

[array.width][array.height][neighbours]CHAN OF STATE link :
[array.width][array.height]CHAN OF COMMAND control :
[array.width][array.height]CHAN OF RESPONSE sense :
PAR
  controller(terminal.keyboard, terminal.screen, control, sense)
  PAR x = 0 FOR array.width
    PAR y = 0 FOR array.height
      VAL INT left  IS ((x - 1) + array.width)  \ array.width  :
      VAL INT right IS  (x + 1)                 \ array.width  :
      VAL INT up    IS  (y + 1)                 \ array.height :
      VAL INT down  IS ((y - 1) + array.height) \ array.height :
      VAL [neighbours]INT nx IS
            [ right, x,    left, left, left, x,  right, right ] :
      VAL [neighbours]INT ny IS
            [ down,  down, down, y,    up,   up, up,    y     ] :
      cell(link, x, y, nx, ny, control[x][y], sense[x][y])
```