

decimal

\ requires random.f

variable seed -345 seed !

```
: random
  seed @ dup 12345678 * over + over 7 / xor - dup seed +!
;
```

\ Neighborhood address lines

```
12 constant /address-lines
create address-lines /address-lines 10 + allot
```

variable address-value

```
: address-line ( line --- addr )
  address-lines +
;

: address-line@ ( line --- b )
  address-line c@
  if 1 else 0 then
;

: address-line! ( b line --- )
  swap if 1 else 0 then swap
  address-line c!
;

: clear-address-lines
  address-lines /address-lines erase
;
```

clear-address-lines

```
: set-address-lines ( n --- )
dup address-value !
  /address-lines 0 do
    dup 1 and i address-line!
    2/
  loop
drop
;
```

\ Associate a name with an address line

```
: == ( n --- ) \ name
  create ,
  does> @ address-line@
;
```

\ Neighborhood definitions

\ Major neighborhoods:

\ n/moore	n/vonn	n/marg	n/marg-ph	n/marg-hv
-----------	--------	--------	-----------	-----------

```

0 == center
1 == center'
2 == s.east      2 == east'      2 == cw
3 == s.west      3 == west'      3 == ccw
4 == n.east      4 == south'     4 == opp
5 == n.west      5 == north'     5 == cw'
6 == east        6 == ccw'
7 == west        7 == opp'
8 == south
9 == north

```

```

8 == phase      8 == horz
9 == phase'     9 == vert

```

```

\ Minor neighborhoods:
\ &/centers      &/phases      &/hv

```

```

10 == &center    10 == &phase    10 == &horz
11 == &center'   11 == &phase'   11 == &vert

```

```

\ Higher level neighborhood stuff

```

```

: centers ( --- n )
  center center' 2* +
;

```

```

: norths ( --- n )
  north north' 2* +
;

```

```

: souths ( --- n )
  south south' 2* +
;

```

```

: easts ( --- n )
  east east' 2* +
;

```

```

: wests ( --- n )
  west west' 2* +
;

```

```

\ Rule table

```

```

: 2^x ( x --- 2^x )
  1 swap 0 ?do
    2*
  loop
;

```

```

/address-lines 2^x constant /rule-table
create rule-table /rule-table 10 + allot

```

```

: table-index
  address-value @
;

```

```

: table@ ( index --- b )
  rule-table + c@
;

```

```

: table! ( b --- )
  rule-table + c!
;

: clear-rule-table
  rule-table /rule-table erase
;

clear-rule-table

: dump-rule-table
  rule-table /rule-table dump
;

\ Plane manipulation

: >planer ( mask shift --- ) \ name
  create c, c,
  does> ( b >body --- )
    >r \ b
    r@ c@ << \ b<<shift
    r@ 1+ c@ and \ bs&!mask
    table-index table@ \ bsm t
    r> 1+ c@ not and \ bsm t&!mask
    or \ bsm|tm
    table-index table! \
;

1 0 >planer >pln0
2 1 >planer >pln1
4 2 >planer >pln2
8 3 >planer >pln3
3 0 >planer >plna
12 2 >planer >plnb
16 4 >planer >aux0
32 5 >planer >aux1
64 6 >planer >aux2
128 7 >planer >aux3
48 4 >planer >auxa
192 6 >planer >auxb

\ Rule compiler

variable rule-function

: compile-rule-function ( rule-function --- )
  rule-function token!
  /address-lines 2^x 0 do
    i set-address-lines \ set neighborhood state
    rule-function token@ execute \ execute rule
  \ i 255 and 0= if ." ." then
  loop
;

: compile-rule \ rule
  ' compile-rule-function
;

```

\ Case statement

```
: ( { ( n --- )
  /token *
  r@ /token + /w +
  + token@ execute
;

: {
  compile ( {
    compile branch >mark
; immediate

: }
  >resolve
; immediate

: binary
  2 base !
;
```

\ Cell matrix

```
256 constant /matrix-edge
/matrix-edge dup * constant /cell-matrix
variable >cell-matrix
variable >result-matrix

: cell-matrix ( --- addr )
  >cell-matrix @
;

: result-matrix ( --- addr )
  >result-matrix @
;

: alloc-cells
  /cell-matrix alloc-mem >cell-matrix !
  /cell-matrix alloc-mem >result-matrix !
;

: cell-addr ( x y --- addr )
  8 << + cell-matrix +
;

: cell@ ( x y --- b )
  cell-addr c@
;

: cell! ( b x y --- b )
  cell-addr c!
;

: fill-matrix ( byte matrix --- )
  /cell-matrix rot fill
;

: clear-matrix ( matrix --- )
```

```

0 swap fill-matrix
;

: randomize-matrix ( matrix --- )
/cell-matrix 0 do
  here random 2/ 2/ 8191 and - c@
  random 2/ 2/ 2/ 2/ xor
  random 2/ xor
  3 and
  over c!
  random 8 and if random drop then
  1+ loop
drop
;

code moore-index ( x y --- index )
>cell-matrix 1#) a0 lmove
sp )+ d1 lmove
8 l# d1 long lsl
sp ) d1 long or
0 d1 a0 di)l d2 bmove      \ center' center
3 l# d2 long and

256 # d1 word sub          \ north
0 d1 a0 di)l d3 byte move
1 # d3 byte and
0<> if
  512 # d2 word or
then

1 # d1 byte sub            \ n.west
0 d1 a0 di)l d3 bmove
1 # d3 byte and
0<> if
  32 # d2 word or
then

256 # d1 word add          \ west
0 d1 a0 di)l d3 bmove
1 # d3 byte and
0<> if
  128 # d2 word or
then

256 # d1 word add          \ s.west
0 d1 a0 di)l d3 bmove
1 # d3 byte and
0<> if
  8 # d2 word or
then

1 # d1 byte add            \ south
0 d1 a0 di)l d3 bmove
1 # d3 byte and
0<> if
  256 # d2 word or
then

```

```

1 # d1 byte add \ s.east
0 d1 a0 di)l d3 bmove
1 # d3 byte and
0<> if
    4 # d2 word or
then

256 # d1 word sub \ east
0 d1 a0 di)l d3 bmove
1 # d3 byte and
0<> if
    64 # d2 word or
then

256 # d1 word sub \ n.east
0 d1 a0 di)l d3 bmove
1 # d3 byte and
0<> if
    16 # d2 word or
then

d2 sp ) long move
c;

: .moore-index ( index --- )
base @ 2 base ! swap
set-address-lines
n.west 3 .r      north 3 .r      n.east 3 .r      cr
west 3 .r        centers 3 .r     east 3 .r        cr
s.west 3 .r      south 3 .r      s.east 3 .r      cr
base !
;

: .mi ( x y --- )
moore-index .moore-index
;

: echo
center >pln1
;

: 8sum ( --- n )
north south east west n.west n.east s.west s.east
+ + + + + + +
;

: 9sum ( --- n )
8sum center +
;

: step ( --- )
result-matrix
/matrix-edge 0 do
    /matrix-edge 0 do
        i j moore-index table@
        over c! 1+
    loop

```

```

loop
drop
cell-matrix result-matrix
>cell-matrix ! >result-matrix !
;

variable ras-width
variable ras-height

: look ( x y w h --- )
base @ >r hex
swap ras-width !
0 do
  ras-width @ 0 do
\    2dup moore-index 4 .r ." : "
    2dup cell@ 15 and
    dup 0 = if
      drop space
    else
      0 .r then
      swap 1+ swap
    loop
    swap width @ - swap 1+
  cr
loop
drop drop
r> base !
;

: hex-type ( addr len --- )
base @ >r hex
0 ?do count 3 .r loop
drop
r> base !
;

create gray-levels
0 c, 85 c, 170 c, 255 c,

: ps-type-pixels ( addr len --- )
0 ?do
  count gray-levels + c@
  ps-emit
loop drop
;

: image-raster ( x y w h --- )
ras-height ! ras-width !
ras-width @ ps-int ras-height @ ps-int 8 ps-int
c~ [~ ps-text
  ras-width @ ps-int 0 ps-int 0 ps-int
  ras-height @ ps-int 0 ps-int 0 ps-int
c~ ]{currentfile ~ ps-text
ras-width @ ras-height @ * ps-int
c~ string readstring pop}image ~ ps-text
ras-height @ 0 do \ x y
  over over cell-addr ras-width @
  ps-type-pixels

```

```

    1+
  loop
  drop drop
;

: update-cam-raster ( x y w h --- )
  c~ gsave(!cam-raster)findraster setcanvas clippath pathbbox scale pop pop~
  ps-text
  image-raster
  c~ grestore /paint PileDict ContentsPileID get /Win get send ~ ps-text
;

variable cam-view-x  cam-view-x off
variable cam-view-y  cam-view-y off
variable cam-view-w  64 cam-view-w !
variable cam-view-h  64 cam-view-h !

: .update-cam
  cam-view-x @ cam-view-y @ cam-view-w @ cam-view-h @
  update-cam-raster
  .f
;

: .step
  step .update-cam
;

```