

Movement of Ants

File: Ants.mw

Based uncharted 12, "Chemotaxis," from *Modeling Nature: Cellular Automata Simulations with Mathematica* by Richard J. Gaylord and Kazume Nishidate, Copyright 1996 TELOS/Springer-Verlag.

Development of Ants Program

Lattice (matrix, grid)

Each lattice site has value of an ordered pair of integers

1st component - level of chemical attractant, nonnegative integer

2nd component -

EMPTY = "T" - empty site

NORTH = "N" - N-facing ant

EAST = "E" - E-facing ant

SOUTH = "S" - S-facing ant

WEST = "W" - W-facing ant

Initially, 0 level of chemical attractant at all sites and random direction.

probAnt = probability of cell being occupied

```
> # constants
EMPTY := "T":
NORTH := "N":
EAST := "E":
SOUTH := "S":
WEST := "W":
```

Update rules

1. Apply rules for orienting ant

sense[site, N, E, S, W]

Empty site remains unchanged

Ant orients itself so as to face nearest neighbor site that has highest chemical attractant level.

If two or more nearest neighbor sites have equal amount, pick at random.

2. Apply rules for diffusing ant on lattice

walk[site, N, E, S, W, NE, SE, SW, NW, Nn, Ee, Ss, Ww]

When ant moves, leaves behind chemical scent to attract other ants. (increase scent value by 1)

In absence of ant, chemical scent that has been laid down dissipates. (Decrease by 1)

Sensing rules

Orientation rule - *sense* function:

Empty site remains unchanged

```
> undefine(sense):
define(sense,
sense([a::integer, EMPTY], b::list, c::list,
d::list, e::list) =
[a, EMPTY]);
```

Orientation rule - *sense* function:

Ant turns to highest scent in N, E, S, W direction: at random to equal maximum locations

First define a function, *senseMax*, that returns the direction, NORTH, EAST, SOUTH, WEST, of the maximum of four arguments, respectively. If more than one argument is equal to the maximum, one is picked at random.

```
> unassign(senseMax):
senseMax := proc(n, e, s, w)
  local mx, mxDirList, lng;
  mx := max(n, e, s, w):
  mxDirList := [ `if` (n = mx, NORTH, NULL),
                `if` (e = mx, EAST, NULL),
                `if` (s = mx, SOUTH, NULL),
                `if` (w = mx, WEST, NULL) ]:
  lng := nops(mxDirList):
  mxDirList[rand(1..lng)()];
end proc;
```

The rule uses delayed evaluation of *senseMax*.

```
> definemore(sense,
  sense([a::integer, aa::character],
        [b::integer, nn::character],
        [e::integer, ee::character],
        [s::integer, ss::character],
        [w::integer, ww::character]) =
  [a, 'senseMax'(b, e, s, w)]
);
```

Walking Rules

Rules for movement of ant - *walk* function:

We must be careful to list the most specific rules first, because *Maple* applies the first rule that it can.

If movement would cause a collision, do not vacate site and do not change amount of chemical at site.

```
> undefine(walk);
> define(walk,
  walk([a::integer, NORTH], [b::integer, EMPTY],
        E::list, S::list, W::list,
        [ne::integer, WEST], SE::list, SW::list,
        NW::list, Nn::list,
        Ee::list, Ss::list, Ww::list) =
  [a, NORTH],

  walk([a::integer, NORTH], [b::integer, EMPTY],
        E::list, S::list, W::list,
        NE::list, SE::list, SW::list,
        [nw::integer, EAST], Nn::list,
        Ee::list, Ss::list, Ww::list) =
  [a, NORTH],

  walk([a::integer, NORTH], [b::integer, EMPTY],
```

```

E::list, S::list, W::list,
NE::list, SE::list, SW::list,
NW::list, [nn::integer, SOUTH],
Ee::list, Ss::list, Ww::list) =
    [a, NORTH],

walk([a::integer, EAST], N::list,
    [e::integer, EMPTY], S::list,
    W::list, [ne::integer, SOUTH], SE::list,
    SW::list, NW::list,
    Nn::list, Ee::list, Ss::list, Ww::list) =
    [a, EAST],

walk([a::integer, EAST], N::list,
    [e::integer, EMPTY], S::list,
    W::list, NE::list, [se::integer, NORTH],
    SW::list, NW::list,
    Nn::list, Ee::list, Ss::list, Ww::list) =
    [a, EAST],

walk([a::integer, EAST], N::list,
    [e::integer, EMPTY], S::list,
    W::list, NE::list, SE::list, SW::list,
    NW::list, Nn::list,
    [ee::integer, WEST], Ss::list, Ww::list) =
    [a, EAST],

walk([a::integer, SOUTH], N::list,
    E::list, [s::integer, EMPTY],
    W::list, NE::list, [se::integer, WEST],
    SW::list, NW::list,
    Nn::list, Ee::list, Ss::list, Ww::list) =
    [a, SOUTH],

walk([a::integer, SOUTH], N::list,
    E::list, [s::integer, EMPTY],
    W::list, NE::list, SE::list,
    [sw::integer, EAST], NW::list,
    Nn::list, Ee::list, Ss::list, Ww::list) =
    [a, SOUTH],

walk([a::integer, SOUTH], N::list,
    E::list, [s::integer, EMPTY],
    W::list, NE::list, SE::list,
    SW::list, NW::list, Nn::list,
    Ee::list, [ss::integer, NORTH], Ww::list) =
    [a, SOUTH],

walk([a::integer, WEST], N::list,
    E::list, S::list, [w::integer, EMPTY],
    NE::list, SE::list, [sw::integer, NORTH],

```

```

NW::list, Nn::list,
Ee::list, Ss::list, Ww::list) =
    [a, WEST],

walk([a::integer, WEST], N::list,
    E::list, S::list, [w::integer, EMPTY],
    NE::list, SE::list, SW::list,
    [nw::integer, SOUTH],
    Nn::list, Ee::list, Ss::list, Ww::list) =
    [a, WEST],

walk([a::integer, WEST], N::list,
    E::list, S::list, [w::integer, EMPTY],
    NE::list, SE::list, SW::list, NW::list,
    Nn::list, Ee::list, Ss::list,
    [ww::integer, EAST]) =
    [a, WEST]
);

```

Have rules for facing each direction.

If move, increase by 1 the amount of chemical in site vacated.

```

> definemore(walk,
    walk([a::integer, NORTH], [b::integer, EMPTY],
        E::list, S::list, W::list, NE::list,
        SE::list, SW::list, NW::list,
        Nn::list, Ee::list, Ss::list, Ww::list) =
            [ `if` (a>0, a+1, 0), EMPTY],

    walk([a::integer, EAST], N::list,
        [e::integer, EMPTY], S::list, W::list,
        NE::list, SE::list, SW::list, NW::list,
        Nn::list, Ee::list, Ss::list, Ww::list) =
            [ `if` (a>0, a+1, 0), EMPTY],

    walk([a::integer, SOUTH], N::list, E::list,
        [s::integer, EMPTY], W::list, NE::list,
        SE::list, SW::list, NW::list,
        Nn::list, Ee::list, Ss::list, Ww::list) =
            [ `if` (a>0, a+1, 0), EMPTY],

    walk([a::integer, WEST], N::list, E::list,
        S::list, [w::integer, EMPTY], NE::list,
        SE::list, SW::list, NW::list,
        Nn::list, Ee::list, Ss::list, Ww::list) =
            [ `if` (a>0, a+1, 0), EMPTY]
);

```

If two ants want to move into site, do not allow either to do so and decrease the value of the chemical scent by 1 to minimum of 0.

```

> definemore(walk,
    walk([a::integer, EMPTY], [b::integer, SOUTH],
        [e::integer, WEST], S::list, W::list,
        NE::list, SE::list, SW::list, NW::list,

```

```

    Nn::list, Ee::list, Ss::list, Ww::list)=
    [max(a - 1, 0), EMPTY],

walk([a::integer, EMPTY], [b::integer, SOUTH],
    E::list, [s::integer, NORTH], W::list,
    NE::list, SE::list, SW::list, NW::list,
    Nn::list, Ee::list, Ss::list, Ww::list) =
    [max(a - 1, 0), EMPTY],

walk([a::integer, EMPTY], [b::integer, SOUTH],
    E::list, S::list, [w::integer, EAST],
    NE::list, SE::list, SW::list, NW::list,
    Nn::list, Ee::list, Ss::list, Ww::list) =
    [max(a - 1, 0), EMPTY],

walk([a::integer, EMPTY], N::list,
    [e::integer, WEST], [s::integer, NORTH],
    W::list, NE::list, SE::list, SW::list,
    NW::list, Nn::list, Ee::list,
    Ss::list, Ww::list) =
    [max(a - 1, 0), EMPTY],

walk([a::integer, EMPTY], N::list,
    [e::integer, WEST], S::list,
    [w::integer, EAST], NE::list, SE::list,
    SW::list, NW::list, Nn::list, Ee::list,
    Ss::list, Ww::list) =
    [max(a - 1, 0), EMPTY],

walk([a::integer, EMPTY], N::list, E::list,
    [s::integer, NORTH], [w::integer, EAST],
    NE::list, SE::list, SW::list, NW::list,
    Nn::list, Ee::list, Ss::list, Ww::list)=
    [max(a - 1, 0), EMPTY]
);

```

If site is not occupied and an ant to the N,E,S,or W wants to move into the site, let it do so. Keep the current level of chemical at site.

```

> definemore(walk,
    walk([a::integer, EMPTY], [b::integer, SOUTH],
        E::list, S::list, W::list,
        NE::list, SE::list, SW::list, NW::list,
        Nn::list, Ee::list,
        Ss::list, Ww::list) = [a, SOUTH],

    walk([a::integer, EMPTY],
        N::list, [e::integer, WEST], S::list,
        W::list, NE::list, SE::list, SW::list,
        NW::list, Nn::list, Ee::list,
        Ss::list, Ww::list) = [a, WEST],

    walk([a::integer,EMPTY], N::list,
        E::list, [s::integer, NORTH], W::list,
        NE::list, SE::list, SW::list, NW::list,

```

```

Nn::list, Ee::list,
Ss::list, Ww::list)=[a, NORTH],

walk([a::integer, EMPTY], N::list,
E::list, S::list, [w::integer, EAST],
NE::list, SE::list,
SW::list, NW::list, Nn::list, Ee::list,
Ss::list, Ww::list) = [a, EAST]
);

```

[If no ant, decrease strength of scent by 1, but not lower than 0.

```

> definemore(walk,
walk([a::integer, EMPTY], N::list, E::list,
S::list, W::list, NE::list, SE::list,
SW::list, NW::list, Nn::list, Ee::list,
Ss::list, Ww::list) = [max(a - 1, 0), EMPTY]
);

```

[For other collisions, stay put with no increase in chemical.

```

> definemore(walk,
walk([a::integer, c::character], N::list,
E::list, S::list, W::list,
NE::list, SE::list, SW::list, NW::list,
Nn::list, Ee::list,
Ss::list, Ww::list) = [a, c]
);

```

— Boundaries

Periodic boundary conditions in this case

Infinite system

Extend boundary by 1 for sense and by 2 ~~for~~ *walk*

Function to return an $(n + 2)$ -by- $(n + 2)$ matrix for periodic boundaries.

mat, n -by- n matrix

```

> unassign(extendLat1):
extendLat1 := proc(mat)
local matNS, trans, transeW;

# glue on wrap of N & S rows
matNS := [mat[-1], op(mat), mat[1]]:

# glue on wrap of E & W columns assuming
# original matrix is square
trans := ListTools[Transpose](matNS):
transeW := [trans[-1], op(trans), trans[1]]:
ListTools[Transpose](transeW);
end proc:

```

Function to return an $(n + 4)$ -by- $(n + 4)$ matrix for periodic boundaries.

mat, n -by- n matrix

```

> unassign(extendLat2):
n := 'n':

```

```

mat := 'mat':

extendLat2 := proc(mat)
  local matNS, trans, transeW;

  # glue on wrap of N & S rows by 2
  matNS := [mat[-2], mat[-1], op(mat), mat[1], mat[2]]:

  # glue on wrap of E & W columns by 2 assuming
  # original matrix is square
  trans := ListTools[Transpose](matNS):
  transeW := [trans[-2], trans[-1], op(trans), trans[1],
trans[2]]:
  ListTools[Transpose](transeW);
end proc:

```

[-] Functions to apply a function parameter to extended matrix

Function to apply a function parameter to a matrix extended by 1 cell in each direction

Function returns a matrix of applications of function *fnc*, called as *fnc(site, N, E, S, W)*, to each element (*site*) of extended matrix *matExt* except for the first and last rows and columns

```

> unassign(applyExtended1):
n := 'n':

applyExtended1 := proc(fnc, matExt)
  local n;
  n := nops(matExt) - 2:
  [seq(
    [seq( fnc(matExt[i, j], matExt[i - 1, j],
matExt[i, j + 1], matExt[i + 1, j],
matExt[i, j - 1]), j = 2..(n + 1) )
    ],
    i = 2..(n + 1) )
];
end proc:

```

Function to apply a function parameter to a matrix extended by two cells in each direction

Function returns a matrix of applications of function *fnc*, called as *fnc(site, N, E, S, W, NE, SE, SW, NW, Nn, Ee, Ss, Ww)*, to each element (*site*) of extended matrix *matExt* except for the first two and last two rows and columns

```

> unassign(applyExtended2):
n := 'n':

applyExtended2 := proc(fnc, matExt)
  local n;
  n := nops(matExt) - 4:
  [seq(
    [seq( fnc(matExt[i, j], matExt[i-1, j],
matExt[i, j+1], matExt[i+1, j],
matExt[i, j-1], matExt[i-1, j+1],
matExt[i+1, j+1], matExt[i+1, j-1],

```

```

        matExt[i-1, j-1], matExt[i-2, j],
        matExt[i, j+2], matExt[i+2, j],
        matExt[i, j-2]), j = 3..(n + 2) )
    ],
    i = 3..(n + 2))
];
end proc:

```

Ants simulation

```

> unassign(ants):
initialGrid := 'initialGrid':
t := 't':

ants := proc(initialGrid, t)
    local aGrid, gridList, emat1, gridSense, emat2, i;

    aGrid := initialGrid:
    gridList := [aGrid]:

    for i from 1 to t do
        emat1 := extendLat1(aGrid):
        gridSense := applyExtended1(sense, emat1):
        emat2 := extendLat2(gridSense):
        aGrid := applyExtended2(walk, emat2):
        gridList := [op(gridList), aGrid];
    end do;

    gridList;
end proc:

```

Animation of simulation

Function to display a list of grids show ant as red color with degree of color depending on amount of chemical in cell: show empty cell with shade of gray

Note that the sequence of squares for a grid are grouped into a display with `plots[display](plotGrid)` to be appended to the end of the list of grid plots, `plotList`.

`maxAmtChem` is a global variable that `ants` must define.

```

> unassign(matchColor):
maxAmtChem := 'maxAmtChem':
mat := 'mat':
i := 'i':
j := 'j':

# Function to associate color with cell's value
matchColor := proc(mat, i, j)
    local siteColorAmt:
    global maxChem:

    siteColorAmt := 1.0 - (mat[i, j, 1] * 0.5)/maxChem:

    if (mat[i, j, 2] = EMPTY) then

```

```

        COLOR(RGB, siteColorAmt, siteColorAmt,
              siteColorAmt);
    else
        COLOR(RGB, siteColorAmt, 0, 0);
    end if;
end proc:

unassign(showGraphs):
# Function to display a list of grids starting
# at 1 through entire list of grids
# EMPTY (0) shows level of gray corresponding to
# amount of chemical
# Non-EMPTY (NORTH = 1, EAST = 2, SOUTH = 3,
# WEST = 4) shows level of red corresponding to
# amount of chemical
#
# Function showGraphs gives value to global
# maxAmtChem for matchColor function.

showGraphs := proc(graphList, maxChem)
    local k, g, aSquare, plotGrid, plotList, n;
    global maxAmtChem:

    maxAmtChem := maxChem:

    plotList := []:
    aSquare := [[0, 0], [0, 1], n[1, 1], [1, 0]]:
    for k from 1 to nops(graphList) do
        g := graphList[k]: # grid at k-th time step
        n := nops(g):
        plotGrid := seq(seq(
            plottools[translate](
                plots[polygonplot](aSquare,
                    axes = none,
                    scaling = CONSTRAINED,
                    color = matchColor(g, i, j)),
                j, i),
            j = 1..n), i = 1..n):
        plotList := [op(plotList),
                    plots[display](plotGrid)]:
    end do;

    plots[display](plotList, insequence = true);
end proc:

```

Run Program

[Initialization of grid

```

> unassign(rand0to1):
   rand0to1 := proc()
       stats[random, uniform]();
   end proc:
> n:=17:

```

```

probAnt := 0.05: # Probability of an ant in a cell
dir := [NORTH,EAST, SOUTH, WEST]:
maxChem := 50:
randomize(9):
initialGrid:=
  [seq([seq([0, `if`(rand0to1() < probAnt,
              dir[rand(1..4)()], EMPTY)],
        j = 1..n)], i = 1..n]):

for j from 1 to n do
  initialGrid[trunc(n/2) + 1, j] :=
    [trunc(maxChem*j/n), EMPTY]
end do:
[ants(initialGrid, t)
> graphList := ants(initialGrid, 5):
> showGraphs(graphList, maxChem);

```

Ant Functions in One Cell

```

> # constants
EMPTY:= "T":
NORTH :="N":
EAST := "E":
SOUTH := "S":
WEST := "W":

##### sense rules #####

# Empty site remains unchanged

undefine(sense):
define(sense,
  sense([a::integer, EMPTY], b::list, c::list,
        d::list, e::list) =
  [a, EMPTY]);

# senseMax returns the direction, NORTH, EAST,
# SOUTH, WEST, of the maximum of four arguments,
# respectively. If more than one argument is equal
# to the maximum, one is picked at random.

unassign(senseMax):
senseMax := proc(n, e, s, w)
  local mx, mxDirList, lng;
  mx := max(n, e, s, w):
  mxDirList := [ `if`(n = mx, NORTH, NULL),
                `if`(e = mx, EAST, NULL),
                `if`(s = mx, SOUTH, NULL),
                `if`(w = mx, WEST, NULL)]:
  lng := nops(mxDirList):
  mxDirList[rand(1..lng)()];

```

```
end proc:
```

```
# Ant turns to highest scent in N,E,S,W direction:  
# at random to equal maximum locations  
# The rule uses delayed evaluation of senseMax.
```

```
definemore(sense,  
  sense([a::integer, aa::character],  
        [b::integer, nn::character],  
        [e::integer, ee::character],  
        [s::integer, ss::character],  
        [w::integer, ww::character]) =  
  [a, 'senseMax'(b, e, s, w)]  
);
```

```
> ##### walk rules #####  
# Rules for movement of ant  
# We must be careful to list the most specific rules  
# first, because Maple applies the first rule that  
# it can.
```

```
# If movement would cause a collision, do not vacate  
# site and do not change amount of chemical at site.
```

```
undefine(walk);
```

```
define(walk,  
  walk([a::integer, NORTH], [b::integer, EMPTY],  
        E::list, S::list, W::list,  
        [ne::integer, WEST], SE::list, SW::list,  
        NW::list, Nn::list,  
        Ee::list, Ss::list, Ww::list) =  
  [a, NORTH],
```

```
  walk([a::integer, NORTH], [b::integer, EMPTY],  
        E::list, S::list, W::list,  
        NE::list, SE::list, SW::list,  
        [nw::integer, EAST], Nn::list,  
        Ee::list, Ss::list, Ww::list) =  
  [a, NORTH],
```

```
  walk([a::integer, NORTH], [b::integer, EMPTY],  
        E::list, S::list, W::list,  
        NE::list, SE::list, SW::list,  
        NW::list, [nn::integer, SOUTH],  
        Ee::list, Ss::list, Ww::list) =  
  [a, NORTH],
```

```
  walk([a::integer, EAST], N::list,  
        [e::integer, EMPTY], S::list,  
        W::list, [ne::integer, SOUTH], SE::list,  
        SW::list, NW::list,  
        Nn::list, Ee::list, Ss::list, Ww::list) =
```

```

    [a, EAST],

walk([a::integer, EAST], N::list,
    [e::integer, EMPTY], S::list,
    W::list, NE::list, [se::integer, NORTH],
    SW::list, NW::list,
    Nn::list, Ee::list, Ss::list, Ww::list) =
    [a, EAST],

walk([a::integer, EAST], N::list,
    [e::integer, EMPTY], S::list,
    W::list, NE::list, SE::list, SW::list,
    NW::list, Nn::list,
    [ee::integer, WEST], Ss::list, Ww::list) =
    [a, EAST],

walk([a::integer, SOUTH], N::list,
    E::list, [s::integer, EMPTY],
    W::list, NE::list, [se::integer, WEST],
    SW::list, NW::list,
    Nn::list, Ee::list, Ss::list, Ww::list) =
    [a, SOUTH],

walk([a::integer, SOUTH], N::list,
    E::list, [s::integer, EMPTY],
    W::list, NE::list, SE::list,
    [sw::integer, EAST], NW::list,
    Nn::list, Ee::list, Ss::list, Ww::list) =
    [a, SOUTH],

walk([a::integer, SOUTH], N::list,
    E::list, [s::integer, EMPTY],
    W::list, NE::list, SE::list,
    SW::list, NW::list, Nn::list,
    Ee::list, [ss::integer, NORTH], Ww::list) =
    [a, SOUTH],

walk([a::integer, WEST], N::list,
    E::list, S::list, [w::integer, EMPTY],
    NE::list, SE::list, [sw::integer, NORTH],
    NW::list, Nn::list,
    Ee::list, Ss::list, Ww::list) =
    [a, WEST],

walk([a::integer, WEST], N::list,
    E::list, S::list, [w::integer, EMPTY],
    NE::list, SE::list, SW::list,
    [nw::integer, SOUTH],
    Nn::list, Ee::list, Ss::list, Ww::list) =
    [a, WEST],

walk([a::integer, WEST], N::list,

```

```

    E::list, S::list, [w::integer, EMPTY],
    NE::list, SE::list, SW::list, NW::list,
    Nn::list, Ee::list, Ss::list,
    [ww::integer, EAST]) =
        [a, WEST]
);

# Have rules for facing each direction.
# If move, increase by 1 the amount of chemical in
# site vacated.

definemore(walk,
    walk([a::integer, NORTH], [b::integer, EMPTY],
        E::list, S::list, W::list, NE::list,
        SE::list, SW::list, NW::list,
        Nn::list, Ee::list, Ss::list, Ww::list) =
        [if(a>0, a+1, 0), EMPTY],

    walk([a::integer, EAST], N::list,
        [e::integer, EMPTY], S::list, W::list,
        NE::list, SE::list, SW::list, NW::list,
        Nn::list, Ee::list, Ss::list, Ww::list) =
        [if(a>0, a+1, 0), EMPTY],

    walk([a::integer, SOUTH], N::list, E::list,
        [s::integer, EMPTY], W::list, NE::list,
        SE::list, SW::list, NW::list,
        Nn::list, Ee::list, Ss::list, Ww::list) =
        [if(a>0, a+1, 0), EMPTY],

    walk([a::integer, WEST], N::list, E::list,
        S::list, [w::integer, EMPTY], NE::list,
        SE::list, SW::list, NW::list,
        Nn::list, Ee::list, Ss::list, Ww::list) =
        [if(a>0, a+1, 0), EMPTY]
);

# If two ants want to move into site, do not allow
# either to do so and decrease the value of the
# chemical scent by 1 to minimum of 0.

definemore(walk,
    walk([a::integer, EMPTY], [b::integer, SOUTH],
        [e::integer, WEST], S::list, W::list,
        NE::list, SE::list, SW::list, NW::list,
        Nn::list, Ee::list, Ss::list, Ww::list)=
        [max(a - 1, 0), EMPTY],

    walk([a::integer, EMPTY], [b::integer, SOUTH],
        E::list, [s::integer, NORTH], W::list,
        NE::list, SE::list, SW::list, NW::list,
        Nn::list, Ee::list, Ss::list, Ww::list) =
        [max(a - 1, 0), EMPTY],

```

```
walk([a::integer, EMPTY], [b::integer, SOUTH],
     E::list, S::list, [w::integer, EAST],
     NE::list, SE::list, SW::list, NW::list,
     Nn::list, Ee::list, Ss::list, Ww::list) =
     [max(a - 1, 0), EMPTY],
```

```
walk([a::integer, EMPTY], N::list,
     [e::integer, WEST], [s::integer, NORTH],
     W::list, NE::list, SE::list, SW::list,
     NW::list, Nn::list, Ee::list,
     Ss::list, Ww::list) =
     [max(a - 1, 0), EMPTY],
```

```
walk([a::integer, EMPTY], N::list,
     [e::integer, WEST], S::list,
     [w::integer, EAST], NE::list, SE::list,
     SW::list, NW::list, Nn::list, Ee::list,
     Ss::list, Ww::list) =
     [max(a - 1, 0), EMPTY],
```

```
walk([a::integer, EMPTY], N::list, E::list,
     [s::integer, NORTH], [w::integer, EAST],
     NE::list, SE::list, SW::list, NW::list,
     Nn::list, Ee::list, Ss::list, Ww::list)=
     [max(a - 1, 0), EMPTY]
```

```
);
```

```
# If site is not occupied and an ant to the N,E,S, or
# W wants to move into the site, let it do so. Keep
# the current level of chemical at site.
```

```
definemore(walk,
  walk([a::integer, EMPTY], [b::integer, SOUTH],
       E::list, S::list, W::list,
       NE::list, SE::list, SW::list, NW::list,
       Nn::list, Ee::list,
       Ss::list, Ww::list) = [a, SOUTH],
```

```
walk([a::integer, EMPTY],
     N::list, [e::integer, WEST], S::list,
     W::list, NE::list, SE::list, SW::list,
     NW::list, Nn::list, Ee::list,
     Ss::list, Ww::list) = [a, WEST],
```

```
walk([a::integer, EMPTY], N::list,
     E::list, [s::integer, NORTH], W::list,
     NE::list, SE::list, SW::list, NW::list,
     Nn::list, Ee::list,
     Ss::list, Ww::list)=[a, NORTH],
```

```
walk([a::integer, EMPTY], N::list,
     E::list, S::list, [w::integer, EAST],
     NE::list, SE::list,
     SW::list, NW::list, Nn::list, Ee::list,
```

```

        Ss::list, Ww::list) = [a, EAST]
);

# If no ant, decrease strength of scent by 1,
# but not lower than 0.

definemore(walk,
  walk([a::integer, EMPTY], N::list, E::list,
    S::list, W::list, NE::list, SE::list,
    SW::list, NW::list, Nn::list, Ee::list,
    Ss::list, Ww::list) = [max(a - 1, 0), EMPTY]
);

# For other collisions, stay put with no increase in
# chemical.

definemore(walk,
  walk([a::integer, c::character], N::list,
    E::list, S::list, W::list,
    NE::list, SE::list, SW::list, NW::list,
    Nn::list, Ee::list,
    Ss::list, Ww::list) = [a, c]
);

##### boundary functions #####

# Function to return an (n + 2)-by-(n + 2) matrix
# for periodic boundaries mat, n-by-n matrix

unassign(extendLat1):
extendLat1 := proc(mat)
  local matNS, trans, transEW;

  # glue on wrap of N & S rows
  matNS := [mat[-1], op(mat), mat[1]]:

  # glue on wrap of E & W columns assuming
  # original matrix is square
  trans := ListTools[Transpose](matNS):
  transEW := [trans[-1], op(trans), trans[1]]:
  ListTools[Transpose](transEW);
end proc:

unassign(extendLat2):
n := 'n':
mat := 'mat':

# Function to return an (n + 4)-by-(n + 4) matrix
# for periodic boundaries.
# mat, n-by-n matrix

extendLat2 := proc(mat)
  local matNS, trans, transEW;

```

```

# glue on wrap of N & S rows by 2
matNS := [mat[-2], mat[-1], op(mat), mat[1],
          mat[2]]:

# glue on wrap of E & W columns by 2 assuming
# original matrix is square
trans := ListTools[Transpose](matNS):
transEW := [trans[-2], trans[-1], op(trans),
            trans[1], trans[2]]:
ListTools[Transpose](transEW);
end proc:

unassign(applyExtended1):
n := 'n':

# Function to apply a function parameter to a
# matrix extended by 1 cell in each direction.
# Function returns a matrix of applications of
# function fnc, called as fnc(site, N, E, S, W),
# to each element (site) of extended matrix matExt
# except for the first and last rows and columns

applyExtended1 := proc(fnc, matExt)
  local n;
  n := nops(matExt) - 2:
  [seq(
    [seq( fnc(matExt[i, j], matExt[i - 1, j],
              matExt[i, j + 1], matExt[i + 1, j],
              matExt[i, j - 1]), j = 2..(n + 1) )
    ],
    i = 2..(n + 1))
  ];
end proc:

# Function to apply a function parameter to a matrix
# extended by two cells in each direction
# Function returns a matrix of applications of
# function fnc, called as fnc(site, N, E, S, W, NE,
# SE, SW, NW, Nn, Ee, Ss, Ww), to each element
# (site) of extended matrix matExt except for the
# first two and last two rows and columns

unassign(applyExtended2):
n := 'n':

applyExtended2 := proc(fnc, matExt)
  local n;
  n := nops(matExt) - 4:
  [seq(
    [seq( fnc(matExt[i, j], matExt[i-1, j],
              matExt[i, j+1], matExt[i+1, j],
              matExt[i, j-1], matExt[i-1, j+1],
              matExt[i+1, j+1], matExt[i+1, j-1],
              matExt[i-1, j-1], matExt[i-2, j],
              matExt[i, j+2], matExt[i, j-2],
              matExt[i+2, j], matExt[i+2, j-1],
              matExt[i-2, j+1], matExt[i-2, j-1],
              matExt[i-2, j-2], matExt[i-1, j+2],
              matExt[i-1, j-2], matExt[i+2, j+1],
              matExt[i+2, j-2], matExt[i-2, j+2],
              matExt[i-2, j-2])
    ],
    i = 2..(n + 2))
  ];
end proc:

```

```

                matExt[i, j+2],    matExt[i+2, j],
                matExt[i, j-2]), j = 3..(n + 2) )
            ],
            i = 3..(n + 2))
        ];
end proc:

##### ants simulation #####

unassign(ants):
initialGrid := 'initialGrid':
t := 't':

ants := proc(initialGrid, t)
    local aGrid, gridList, emat1, gridSense, emat2, i;

    aGrid := initialGrid:
    gridList := [aGrid]:

    for i from 1 to t do
        emat1 := extendLat1(aGrid):
        gridSense := applyExtended1(sense, emat1):
        emat2 := extendLat2(gridSense):
        aGrid := applyExtended2(walk, emat2):
        gridList := [op(gridList), aGrid];
    end do;

    gridList;
end proc:

# Function rand0to1() returns a random floating
# point number between 0 and 1. Function fire and
# spread rules use this function.

unassign(rand0to1):
rand0to1 := proc()
    stats[random, uniform]();
end proc:

# Function to display a list of grids starting
# at 1 through entire list of grids
# Function to display a list of grids show ant as
# red color with degree of color depending on amount
# of chemical in cell: show empty cell with shade of
# gray
# maxAmtChem is a global variable that ants must
# define.

unassign(matchColor):
maxAmtChem := 'maxAmtChem':
mat := 'mat':
i := 'i':

```

```

j := 'j':

# Function to associate color with cell's value
matchColor := proc(mat, i, j)
  local siteColorAmt:
  global maxChem:

  siteColorAmt := 1.0-(mat[i,j,1] * 0.5)/maxChem:

  if (mat[i,j,2] = EMPTY) then
    COLOR(RGB, siteColorAmt, siteColorAmt,
          siteColorAmt);
  else
    COLOR(RGB, siteColorAmt, 0, 0);
  end if;
end proc:

unassign(showGraphs):
# Function to display a list of grids starting
# at 1 through entire list of grids
# EMPTY (0) shows level of gray corresponding to
# amount of chemical
# Non-EMPTY (NORTH = 1, EAST = 2, SOUTH = 3,
# WEST = 4) shows level of red corresponding to
# amount of chemical
#
# Function showGraphs gives value to global
# maxAmtChem for matchColor function.

showGraphs := proc(graphList, maxChem)
  local k, g, aSquare, plotGrid, plotList, n;
  global maxAmtChem:

  maxAmtChem := maxChem:

  plotList := []:
  aSquare := [[0, 0], [0, 1],[1, 1],[1, 0]]:
  for k from 1 to nops(graphList) do
    g := graphList[k]: # grid at k-th time step
    n := nops(g):
    plotGrid := seq(seq(
      plottools[translate](
        plots[polygonplot](aSquare,
          axes = none,
          scaling = CONSTRAINED,
          color = matchColor(g, i, j)),
      j, i),
      j = 1..n), i = 1..n):
    plotList := [op(plotList),
      plots[display](plotGrid)]:
  end do;

  plots[display](plotList, insequence = true);
end proc:

```

```
> ##### Initialization of grid #####  
n:=17:  
probAnt := 0.2: # Probability of an ant in a cell  
dir := [NORTH, EAST, SOUTH, WEST]:  
maxChem := 50:  
randomize(9):  
  
initialGrid:=  
  [seq([seq([0, `if`(rand0to1() < probAnt,  
              dir[rand(1..4)()], EMPTY)],  
        j = 1..n)], i = 1..n):  
  
for j from 1 to n do  
  initialGrid[trunc(n/2) + 1, j] :=  
    [trunc(maxChem*j/n), EMPTY]  
end do:  
  
> graphList := ants(initialGrid, 20):  
> showGraphs(graphList, maxChem);
```


