Initialize the following parameters:

a,p,s: 3 structs (or classes) for containing information about general parameters, ant parameters and values, and spaces and results, respectively.

p.time: time for which the simulation will run

p.grid: grid size

p.plot: container for plot information at each time step; this will be a cell array of size (3,1), with one cell each for food plot, trail plot, ant plot.

p.nest: any of 'center', 'corner', 'random'

p.nest_range: area around the nest where no food will be distributed (in number of grid points)

p.food: array containing food particles and their weights

a.num: number of ants currently in the field (can increase during simulation)

a.max: maximum number of ants that can be present in the field

a.expl_perc: percentage of ants which are exploring

a.mem: memory strength of ants (number of steps that ants try to avoid visiting again)

a.reinf: strength by which the ants reinforce the pheromone trail

a.evap: evaporation rate of pheromone

a.pos: matrix of shape 'a.mem x a.num', where each column represents the positions of a single ant at the corresponding time steps

a.foraging: vector of length 'a.num' containing a bool of whether the ants are foraging or returning with food

s.foodspace: matrix of shape 'p.grid' containing locations and weights of food particles (eg., if there is a food particle of weight 15 at (3,4), then s.foodspace[3,4]=15) s.trailspace: matrix of shape ‘p.grid’ containing locations and weights of pheromone trails

s.food_collected: vector of size ‘p.time’ containing total food weight collected over time

Sample parameters:

p.time = 1000

p.grid = 10 x 10

p.plot = empty cell of shape 3 x 1

p.nest = ‘center’

p.nest_range = 1

p.food = [10,5,2,4]

a.num = 10

a.max = 20

a.expl_perc = 10

a.mem = 20

a.reinf = 2

a.evap = 0.05

a.pos = empty array of shape 20 x 10

s.foodspace = empty array of shape 10 x 10

s.trailspace = empty array of shape 10 x 10

s.food_collected = empty vector of length 1000

Algorithm:

1. *Initialize p.nest and plot it*

   if p.nest == ‘center’, then p.nest = integer((p.grid-1)/2) elif p.nest == ‘corner’, then p.nest = choose randomly from {(0,0), (0,p.grid[1]), (p.grid[0],0), p.grid}

elif p.nest == ‘random’, then p.nest = choose a random point in area

Here, p.nest = (4,4)

plot(p.nest)

2. *Initialize food*

   space = (0:p.grid[0], 0:p.grid[1]) *# list of all coordinates in area*

   space -= (p.nest, get_neighbors(p.nest,p.nest_range)) *# remove coordinates corresponding to the nest and its surrounding area*

   cords = choose randomly ‘length(p.food)’ points from space

   s.foodspace(coords) = p.food

3. *Initialize exploring ants*

   expl_num = round(a.num*a.expl_perc / 100)

   a.expl = choose expl_num ants randomly from 1:a.num

   Here, a.expl = 4, say. Meaning the ant with index 4 is an exploring ant.

4. *Initialize ants at nest*

   a.pos[0,:] = p.nest

5. *Start simulation*

   for t = 1 to p.time

     for i = 0 to a.num – 1

```
    steps = get_neighbors( a.pos[mod(t-1,a.mem),i] )
        if a.foraging[i] == True:
            if any(s.foodspace[steps] != 0), then found_food(a, s, steps, i)
            elif any(s.trailspace[steps] != 0), then found_trail(a, p, s, steps, i)
            else walk(a, p, steps, i)
        else:
            return_to_nest(a, p, s, steps, i)
    s.trailspace = max(0,s.trailspace-a.evap)
```

6. *Plot*

```
delete(p.plot[0])
p.plot[0] = plot(s.foodspace[s.foodspace!=0])
delete(p.plot[1])
p.plot[1] = plot(s.trailspace[s.trailspace!=0])
delete(p.plot[2])
p.plot[2] = plot(a.pos[mod(t+1,a.mem),:])
```

7. *Add new ants at random times*

```
if a.num < a.max:
  a.num += 1
  a.pos = [ a.pos, (zeros(a.mem,1)) ]
  a.pos[0:mod(t+1,a.mem),-1] = p.nest
  if length(a.expl) < round( a.num*a.expl_perc / 100 ):
     a.expl.append(a.num)
```

```
function found_food(a, s, steps, i)
   steps = steps[s.foodspace[steps]!=0]
   if length(steps) > 1, then steps = steps[0]
```

```
    a.pos[mod(t,a.mem),i] = steps
    s.foodspace[steps] -= 1
    a.foraging[i] = False
    return

function found_trail(a, p, s, steps, i)
    steps = find(s.trailspace[steps]!=0)
    for j = 0 : length(steps)
        if (dist(steps[j],p.nest) < dist(a.pos[mod(t-1,a.mem),i],p.nest) && \
        a.foraging[i]) || \
        (dist(steps[j],p.nest) > dist(a.pos[mod(t-1,a.mem),i],p.nest) && \
        !a.foraging[i]), then steps[j] = 0
        if ismember(steps[j],a.pos[:,i]) && a.expl==i, then steps[j] = 0
    next_step = choose randomly from steps considering the values in steps as
    weights
    a.pos[mod(t,a.mem),i] = next_step
    if !a.foraging[i], then s.trailspace[next_step] += a.reinf
    return

function walk(a, p, steps, i)
    current_coords = a.pos[mod(t-1,a.mem),i]
    if t != 1:
        last_coords = a.pos[mod(t-2,a.mem),i]
```

```
        ang = tan_inverse( (current_coords[1] - last_coords[1]) /
        (current_coords[0] - last_coords[0]) )
    else:
        ang = 0
    angvec = [0,45,90,135,-180,-135,-90,-45]
    ang = mod(ang+180,360) – 180
    weights = normpdf(angvec*pi/180, mu=0, sigma=0.5)
    for j = 0 : length(angvec) - 1
        if current_coord[0] + cos(ang+angvec[j]) > p.grid[0] || \
          current_coord[1] + sin(ang+angvec[j]) > p.grid[1] || \
          current_coord[0] + cos(ang+angvec[j]) < 0 || \
          current_coord[1] + cos(ang+angvec[j]) < 0 :
            weights[j] = 0
    next_step = choose randomly from steps based on probabilities in weights
    a.pos[mod(t,a.mem),i] = next_step
    return


function return_to_nest(a, p, s, steps, i)
    if any(steps == p.nest):
        a.pos[mod(t,a.mem),i] = p.nest
        s.food_collected += 1
        a.foraging[i] = True
    elif any(s.trailspace[steps] != 0):
        found_trail(a, p, s, steps, i)
```

```
    else:
        d = []
        for j = 0 : length(steps) – 1
            d.append(dist(steps[j],p.nest))
        [min_d, idx] = min(d)
        a.pos[mod(t,a.mem),i] = steps[idx]
        s.trailspace[steps[idx]] += a.reinf
    return

function coords = get_neighbors(current_coord, range=1)
‘’’ Get all the coordinates within a ‘range’ distance ‘’’
    coords = []
    for i = -range : range
        for j = -range : range
            coords.append( (current_ coord [0]+i, current_ coord [1]+j) )
    coords -= current_ coord
    return coords

function d = dist(coord1, coord2)
‘’’ Get Chebyshev distance between two coordinates ‘’’
    d = max( abs(coord1[0] – coord2[0]), abs(coord1[1] – coord2[1]) )
    return d
```