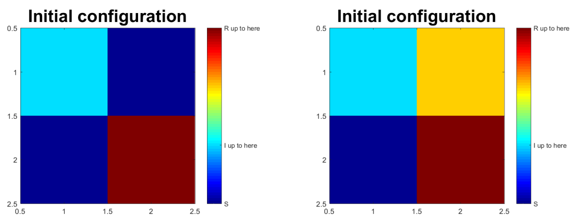
Homework 3.(a)

Modeling Complex Systems, Jack Houk & Javier Lobato

Due date: Thursday, March 8, 2018

A Figure 6.6 of the book

In the next figure, only the initial configuration is shown for each one of the cases shown in Figure 6.6 of the book. These are the `case 1` and `case 2` initial conditions that will be used for the following set of simulations.

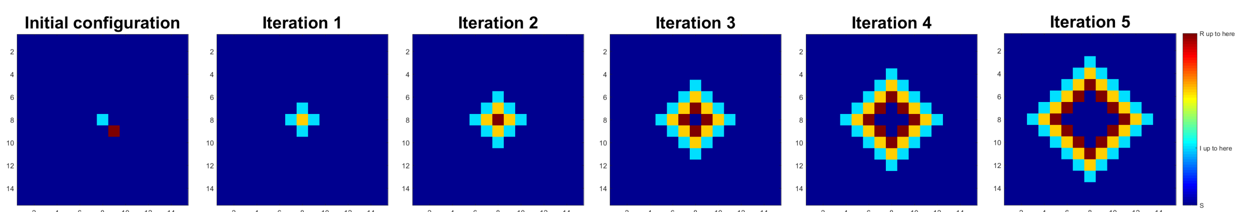


Case (a) Case (b)

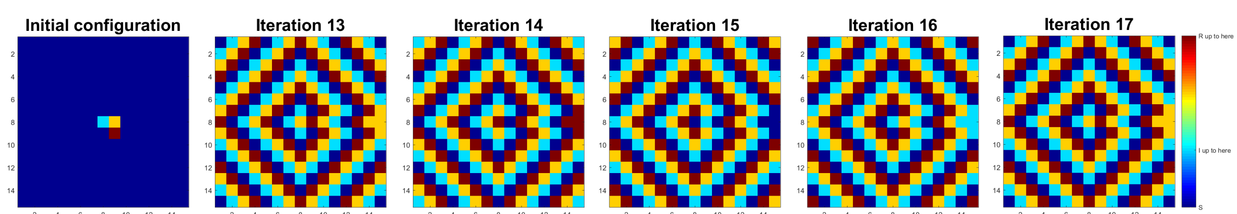
Figure 6.6

B Figure 6.7 of the book

The next simulations try to imitate the Figure 6.7 of the book. They are a deterministic and synchronous simulation with the initial conditions of `case 1` for the upper row and the IC of `case 2` for the second row. The values used are the ones given in the book: $dim = 15$, $a = 1$ and $g = 2$. Only the most representative iteration steps are shown and the color map at the right is the same for all the pictures



Iter. 0 Iter. 1 Iter. 2 Iter. 3 Iter. 4 Iter. 5



Iter. 0 Iter. 13 Iter. 14 Iter. 15 Iter. 16 Iter. 17

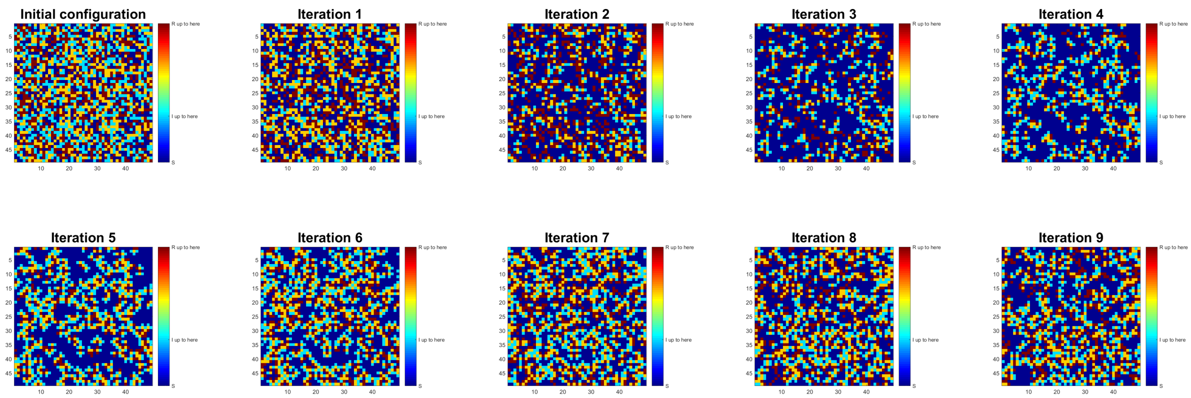
Figure 6.7

C Stochastic simulations

The stochastic simulations try to include some randomness in the simulations, having a probability in the input that will determine if some cell in the state S gets infected or not. Two possible values of that probability are shown below.

High infection probability

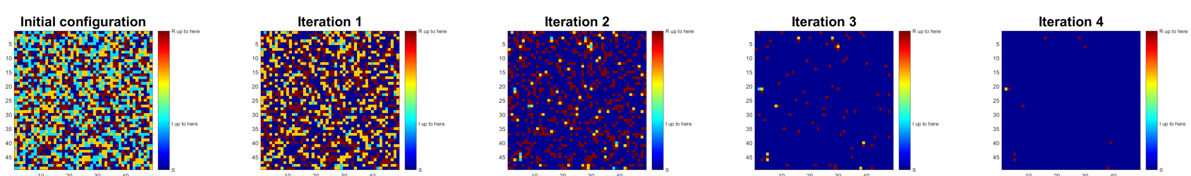
For a relatively high infection probability, the results don't differ too much from the results of the deterministic simulation (which has a $p = 1$). The stochastic, synchronous simulation has been carried out with $dim = 49$, $a = 1$, $g = 2$, $p = 0.7$. The images are ordered from left to right and then from top to bottom:



Stochastic simulation with high infection probability

Low infection probability

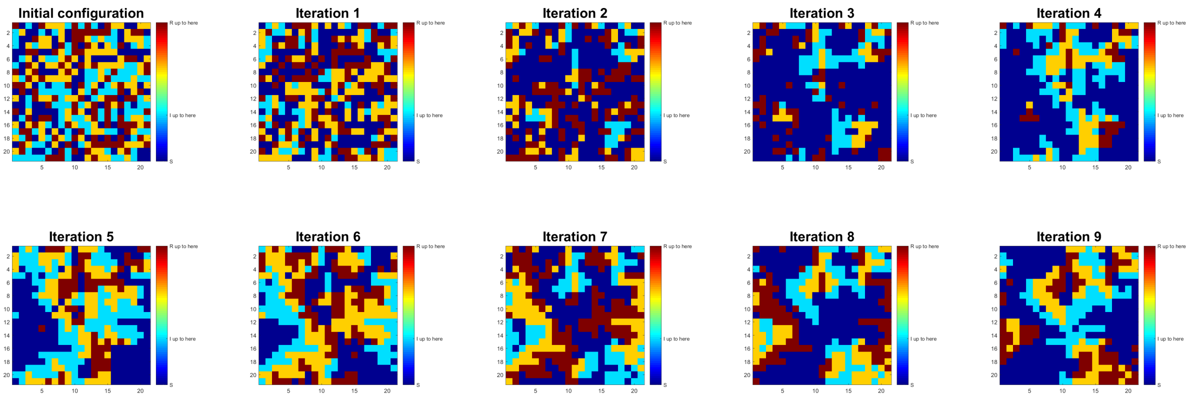
If a low infection probability is imposed, the number of cells that will get infected after recovered may be small - having eventually that the disease just disappear. It can be seen in the set below that the stochastic, synchronous simulation with $dim = 49$, $a = 1$, $g = 2$, $p = 0.1$ the number of infected cases goes to zero in just 5 iterations.



Stochastic simulation with low infection probability

D Asynchronous simulation

Using again a deterministic simulation with $p = 1$, $dim = 21$, $a = 1$, and $g = 2$, but asynchronous update (where the update is done referred to the most recent map instead to the map of the previous iteration in a random order fashion), the results are (images are shown from left to right and then top to bottom):

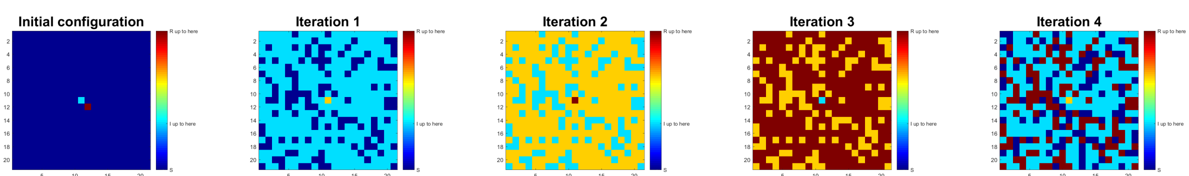


Asynchronous simulation

E Immigration simulation

Finally, to include an immigration rate where susceptible individuals got infected without the need of neighbors to be infected, another percentage was included. Immigration was modeled as a post-infection step where any remaining susceptible individuals had a chance of becoming infected equal to the immigration probability. This model was chosen for two reasons. First, since our grid mapping is toroidal there is no meaningful "edge" for immigration to occur at. Second, it is computationally expedient, handling immigration at the cell level foregoes the need to calculate a global immigration rate at each step.

The percentage has been set to zero in all previous simulations but in this case, the value is $p_I = 0.75$. The other values used are $dim = 21$, $a = 1$, and $g = 2$. The initial conditions are the ones from `case 1` because this way the random infection of cells can be seen more clearly.



Simulation with an immigration rate

Used code

hw3ADriver.m file

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                               HOMEWORK #3.A                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Jack Houk and Javier Lobato, created 03/03/2018
clear all; clc % Let's clear the environment completely

%% Deterministic, synchronous, specified IC (fig 6.6)
%fig 6.6 (a)
greenbergHastings(1, 1, 2, 0, 'case 1', 1, 'sync', 0, 'noSave');
%fig 6.6 (b)
greenbergHastings(1, 1, 2, 0, 'case 2', 1, 'sync', 0, 'noSave');

%% Deterministic, synchronous, IC case 1, longer runs (fig 6.7)
%fig 6.7 (a) - iteration 0
greenbergHastings(15, 1, 2, 0, 'case 1', 1, 'sync', 0, 'saveLast');
%fig 6.7 (a) - iteration 1
greenbergHastings(15, 1, 2, 1, 'case 1', 1, 'sync', 0, 'saveLast');
%fig 6.7 (a) - iteration 2
greenbergHastings(15, 1, 2, 2, 'case 1', 1, 'sync', 0, 'saveLast');
%fig 6.7 (a) - iteration 3
greenbergHastings(15, 1, 2, 3, 'case 1', 1, 'sync', 0, 'saveLast');
%fig 6.7 (a) - iteration 4
greenbergHastings(15, 1, 2, 4, 'case 1', 1, 'sync', 0, 'saveLast');
%fig 6.7 (a) - iteration 5
greenbergHastings(15, 1, 2, 5, 'case 1', 1, 'sync', 0, 'saveLast');

%% Deterministic, synchronous, IC case 1, longer runs (fig 6.7)
%fig 6.7 (b) - iteration 0
greenbergHastings(15, 1, 2, 0, 'case 2', 1, 'sync', 0, 'saveLast');
%fig 6.7 (b) - iteration 13
greenbergHastings(15, 1, 2, 13, 'case 2', 1, 'sync', 0, 'saveLast');
%fig 6.7 (b) - iteration 14
greenbergHastings(15, 1, 2, 14, 'case 2', 1, 'sync', 0, 'saveLast');
%fig 6.7 (b) - iteration 15
greenbergHastings(15, 1, 2, 15, 'case 2', 1, 'sync', 0, 'saveLast');
%fig 6.7 (b) - iteration 16
greenbergHastings(15, 1, 2, 16, 'case 2', 1, 'sync', 0, 'saveLast');
%fig 6.7 (b) - iteration 17
greenbergHastings(15, 1, 2, 17, 'case 2', 1, 'sync', 0, 'saveLast');

%% Stochastic (high infection prob), synchronous, random IC
greenbergHastings(49, 1, 2, 9, 'rand', 0.7, 'sync', 0, 'noSave');

%% Stochastic (low infection prob), synchronous, random IC
greenbergHastings(49, 1, 2, 5, 'rand', 0.1, 'sync', 0, 'noSave');

%% Deterministic, asynchronous, random IC
greenbergHastings(21, 1, 2, 9, 'rand', 1, 'async', 0, 'saveAll');

%% Deterministic, synchronous, IC case 1 to see random immigration
SIR = greenbergHastings(21,1, 2, 4, 'case 1', 1, 'sync', 0.75, 'noSave');
```

`greenbergHastings.m` function

```matlab
function [SIR] = greenbergHastings(dim, a, g, maxt, initializationConf, infectionProb,
↪  sync_async, immigrationRate, saveFigOpt)
% greenbergHastings(dim, a, g, maxt, IC, infectionProb, syncOpt, immgrtRate, saveFigOpt)
% Implement SIR model in a Cellular Automata
% based on: J. M. Greenberg and S. P. Hastings, "Spatial Patterns for
% Discrete Models of Diffusion in Excitable Media", SIAM J. Appl. Math., 34:515-523, 1978
%
% MANDATORY INPUTS:
% dim: dimensions of the square map (same dimension for both sides)
% a: duration of infection in an individual
% g: duration of immunity in an individual
% maxt: maximum number of timesteps to run (user can abort early)
% initializationConf: the implemented initial maps are 'case 1' case 2', or
%       a 'random' initialization of the map at iteration = 0
% infectionProb: the probability of infection of a susceptible cell when
%       surrounded by infected cells. If it is 1 the simulation will be
%       deterministic while if infectionProb < 1, a RNG will determine if a
%       cell is infected or not
% sync_async: option with values 'sync' or 'async' to determine the type of
%       map update
% immigrationRate: probability that susceptible cells go randomly infected.
%       If the value is 0, there will not be any susceptible that gets
%       infected if it is not surrounded by those infected cells
% saveFigOpt: where options are 'saveAll', 'noSave' or 'saveLast' (the last
%       case will only plot the last figure, not showing the others)
%
% OUTPUTS:
% SIR: a 3-column Matrix with the count of susceptible, infected and
% recovered cells at each timestep[nS,nI,nR]
%
% HIGH-LEVEL ABSTRACT STATES
% S = Susceptible to infection
% I = Infected or infectious
% R = Recovered from infection
%
% TRANSITION RULES:
% A cell will remain I for exactly a timesteps, becoming then R
% A cell will remain R for exactly g timesteps, becoming then S
% With probability p, an I cell can infect a neighbor that is S with an
%       overall prob of 1-(1-p)^n, if it has n neighbors that are I
%
% LOW-LEVEL STATES FOR IMPLEMENTATION
% Although the high-level conceptual states are S, I, and R, we will
% actually implement low-level integer states in the range {0,1,,a+g)
% The high-level states will then be inferred as follows:
%    Value of 0 is interpreted as S
%    Values from 1 to a are interpreted as I
%    Values from a+1 to a+g are interpreted as R
% Using this implementation, one can simply increment the low-level values
% for I and R at each timestep, and the value for S when it gets infected
%
% INTERACTION TOPOLOGY:
% The grid is rectangular with Von Neuman neighborhoods and toroidal BC
%
```

```
% SYNCHRONOUS VERSION:
%    Every timestep, update every cell based on values at previous timestep
% ASYNCHRONOUS VERSION:
%    Every timestep, update every cell randomly based on most current values
%

% STUB MADE BY Maggie Eppstein, 2/24/17
%       This stub merely unburdens you from having to figure out how to do
%       the plotting efficiently
% MODIFIED BY Jack Houk and Javier Lobato, 03/03/2018

% Initialize the map with one of the two given cases or with a random map
if strcmp(initializationConf, 'case 1')
    map = zeros(dim);
    map(floor(dim/2)+1, floor(dim/2)+1) = 1;
    map(floor(dim/2)+2, floor(dim/2)+2) = 3;
elseif strcmp(initializationConf, 'case 2')
    map = zeros(dim);
    map(floor(dim/2)+1, floor(dim/2)+1) = 1;
    map(floor(dim/2)+1, floor(dim/2)+2) = 2;
    map(floor(dim/2)+2, floor(dim/2)+2) = 3;
else
    %Random map initialization
    map = randi([0 a+g], dim, dim);
end

% Let's use more representative variable name
duration = a;
recovery = g;

% Preallocation of the output matrix
SIR = zeros([3, maxt+1]);

% Unless only the last map is desired, plot the initial conditions
if ~strcmp(saveFigOpt, 'saveLast')
    [fighandle,plothandle] = plotMapInNewFigure(map,a,g); % Function below
    title('Initial configuration','FontSize', 24)
    pause(0.5) % Pause to see the map
end

% If all iterations are wanted, save the initial contidions as 'it0.png'
if strcmp(saveFigOpt, 'saveAll')
    saveas(gcf,'it0.png')
% Otherwise, if the last one is wanted AND the maximum time is 0, it will
% be saved (if the maximum time is not zero, it is not the last one)
elseif strcmp(saveFigOpt, 'saveLast') && maxt == 0
    [fighandle,plothandle] = plotMapInNewFigure(map,a,g);
    title('Initial configuration','FontSize', 24)
    saveas(gcf,'it0.png')
end

% Store the number of susceptible, infected and recovered
SIR(1, 1) = sum(sum(map == 0));
SIR(2, 1) = sum(sum(map(1 <= map) <= duration));
SIR(3, 1) = sum(sum(duration < map));


```

```
% Synchronous updating case
if strcmp(sync_async, 'sync')
    % Preallocation of a whole new map to be filled
    newMap = zeros(dim);
    % Loop every iteration time step
    for t=1:maxt
        % Loop over the whole map in X and Y
        for x = 1:dim
            for y = 1:dim
                % Get the four neighbours of the current cell (x,y)
                upperNeighbor = map(x, abs(mod(y, dim))+1);
                lowerNeighbor = map(x, abs(mod(y-2, dim))+1);
                leftNeighbor = map(abs(mod(x-2, dim))+1,y);
                rightNeighbor = map(abs(mod(x, dim))+1,y);
                % Store the neighbors in a vector
                neighbors = [upperNeighbor, lowerNeighbor, leftNeighbor, rightNeighbor];
                % With current cell and neighbor value (apart from other
                % parameters such as the probability of infection, the
                % duration of infection, the duration of recovery, and the
                % immigration rate) the value of the new cell will be
                % computed and stored in the same position in the newMap
                newMap(x,y) = cellUpdate(map(x,y), neighbors, infectionProb, duration, recovery, immigrationRate);
            end
        end
        % Reassign the newMap to the old variable map
        map = newMap;
        % Store the number of susceptible, infected and recovered
        SIR(1, t+1) = sum(sum(map == 0));
        SIR(2, t+1) = sum(sum(map(1 <= map) <= duration));
        SIR(3, t+1) = sum(sum(duration < map));

        % Plot the new map if desired (unless saveFigOpt == 'saveLast')
        if ~strcmp(saveFigOpt, 'saveLast')
            % MUCH faster option than doing a new pcolor!
            set(plothandle,'cdata',map);
            % Force matlab to show the figure
            figure(gcf),drawnow;
            % Include the title with the current iteration
            title(['Iteration ' int2str(t)],'FontSize', 24)
            pause(0.5) % Pause to see the map
        end

        % If all iterations are wanted, save the current plot
        if strcmp(saveFigOpt, 'saveAll')
            saveas(gcf,['it' int2str(t) '.png'])
        end
        % If only the last map is wanted (saveFigOpt == 'saveLast') and the
        % current iteration time is the maximum one, save the figure
        if t == maxt && strcmp(saveFigOpt, 'saveLast')
            plotMapInNewFigure(map,a,g);
            title(['Iteration ' int2str(t)],'FontSize', 24)
            saveas(gcf,['it' int2str(maxt) '.png'])
        end
       % Bail out if the user closed the figure (if only the last iteration
       % is wanted, fighandle will not exist
        if ~strcmp(saveFigOpt, 'saveLast')
```

```
            if ~ishandle(fighandle)
                % Plot the final map and exit the loop
                plotMapInNewFigure(map,a,g);
                title('Final configuration','FontSize', 24)
                % Save the last map either if it is the only one to be
                % saved or if all maps have been saved
                if strcmp(saveFigOpt, 'saveAll') || strcmp(saveFigOpt, 'saveLast')
                    saveas(gcf,['it' int2str(maxt) '.png'])
                end
                break
            end
        end
    end

% Asynchronous updating case
else
    % Loop every iteration time step
    for t=1:maxt
        % A random order or substitution will be determined with randperm
        % that gives a random set of number, using the one-entry mode for a
        % 2D array
        for i = randperm(dim^2)
            % Converting the one-entry mode for an array into the classical
            % x and y values for a 2D array
            x = 1 + floor((i-1)/dim);
            y = mod((i-1), dim) + 1;
            % Get the four neighbours of the current cell (x,y)
            upperNeighbor = map(x, abs(mod(y, dim))+1);
            lowerNeighbor = map(x, abs(mod(y-2, dim))+1);
            leftNeighbor = map(abs(mod(x-2, dim))+1,y);
            rightNeighbor = map(abs(mod(x, dim))+1,y);
            % Store the neighbors in a vector
            neighbors = [upperNeighbor, lowerNeighbor, leftNeighbor, rightNeighbor];
            % With current cell and neighbor value (apart from other
            % parameters such as the probability of infection, the duration
            % of infection, the duration of recovery, and the immigration
            % rate) the value of the new cell will be computed and stored
            % in the same position in the SAME map, so the next value of i
            % will have that updated value
            map(x,y) = cellUpdate(map(x,y), neighbors, infectionProb, duration, recovery, immigrationRate);
        end
        % Store the number of susceptible, infected and recovered
        SIR(1, t+1) = sum(sum(map == 0));
        SIR(2, t+1) = sum(sum(map(1 <= map) <= duration));
        SIR(3, t+1) = sum(sum(duration < map));

        % Plot the new map if desired (unless saveFigOpt == 'saveLast')
        if ~strcmp(saveFigOpt, 'saveLast')
            % MUCH faster option than doing a new pcolor!
            set(plothandle,'cdata',map);
            % Force matlab to show the figure
            figure(gcf),drawnow;
            % Include the title with the current iteration
            title(['Iteration ' int2str(t)],'FontSize', 24)
            pause(0.5) % Pause to see the map
        end
```

```
        % If all iterations are wanted, save the current plot
        if strcmp(saveFigOpt, 'saveAll')
            saveas(gcf,['it' int2str(t) '.png'])
        end
        % If only the last map is wanted (saveFigOpt == 'saveLast') and the
        % current iteration time is the maximum one, save the figure
        if t == maxt && strcmp(saveFigOpt, 'saveLast')
            plotMapInNewFigure(map,a,g);
            title(['Iteration ' int2str(t)],'FontSize', 24)
            saveas(gcf,['it' int2str(maxt) '.png'])
        end
       % Bail out if the user closed the figure (if only the last iteration
       % is wanted, fighandle will not exist
        if ~strcmp(saveFigOpt, 'saveLast')
            if ~ishandle(fighandle)
                % Plot the final map and exit the loop
                plotMapInNewFigure(map,a,g);
                title('Final configuration','FontSize', 24)
                % Save the last map either if it is the only one to be
                % saved or if all maps have been saved
                if strcmp(saveFigOpt, 'saveAll') || strcmp(saveFigOpt, 'saveLast')
                    saveas(gcf,['it' int2str(maxt) '.png'])
                end
                break
            end
        end

    end
end
end



% The next function will update the value of one target cell, depending on
% the neighbors that cell have, the probability of infection, duration of
% infection, duration of recovery and a possible immigration rate
 function newState = cellUpdate(target, neighbors, infectionProb, duration, recovery,
 ↪  immigrationRate)
    % If target cell is susceptible
    if target == 0
        % A count of the infected neighbors will be performed along the
        % neigbors input vector
        infectedNeighbors = 0;
        for neighbor = neighbors
            if neighbor > 0 && neighbor <= duration
                infectedNeighbors = infectedNeighbors + 1;
            end
        end
        % If there are neighbours surrounding a susceptible kind of cell,
        % it will get infected with some probability
        if rand < 1-(1-infectionProb)^infectedNeighbors
            newState = 1;
        % If it is not surrounded by infected cells, the susceptible cells
        % will keep as susceptible
        else
            newState = 0;
        end
```

```
    % If target cell is either infected or recovered
    else
        % The value of the cell will increase in 1 for both cases
        newState = target + 1;
        % If the value exceedes the infection and recovery times, the cell
        % will be turned back into the susceptible type
        if newState == duration + recovery + 1
            newState = 0;
        end
    end
    % If a cell is susceptible and exists an immigration rate
    if newState == 0 && rand < immigrationRate
        % It might get infected randomly
        newState = 1;
    end
 end

% Function to plot the map using imagesc()
function [fighandle,plothandle] = plotMapInNewFigure(map, a, g)
    fighandle = figure;
    % Specify location of figure
    set(fighandle,'position',[42   256   560   420]);
    % Doesn't truncate a row and column like pcolor does
    plothandle = imagesc(map);
    colormap(jet);
    % Make sure the color limits don't change dynamically
    set(gca,'clim',[0 a+g]);
    ch = colorbar;
    set(ch,'Ytick',[0 a a+g],'Yticklabel',{'S','I up to here','R up to here'})
    % Make sure aspect ratio is equal
    axis('square')
end
```