Project 1

Shad Mahmod
Modelling Complex Systems

April 4, 2018

1 A firing brain

In this part a model of cellular automata was implemented on a NxN grid on which the boundaries were periodic, e.g. the rightmost end of the grid connected to the leftmost grid.

1.1

In fig.1 we see the inital mesh. The colors {blue, green, yellow} correspond to the states {ready, firing, resting} in the corresponding order. In fig.2 the state of the grid at the iterations {10, 20, 100, 1000} are shown.

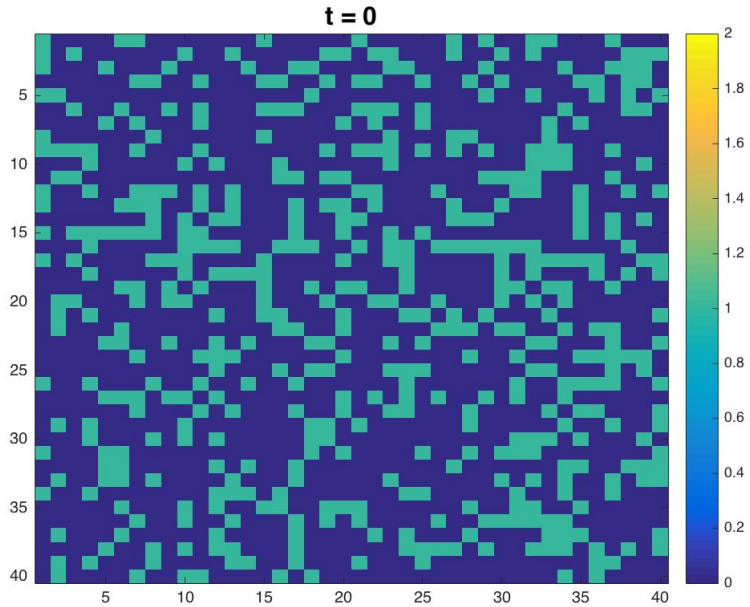


Figure 1: The initial state of the cells

Even though the initial state of the system consists of *firing*, after only then iterations this number is significantly reduced. At $t = 10$ the system have started to converge towards some patches of firing and resting cells. At $t = \{20, 100, 1000\}$ the trend seems to continue where the patches, or groups, of cells are reduced.

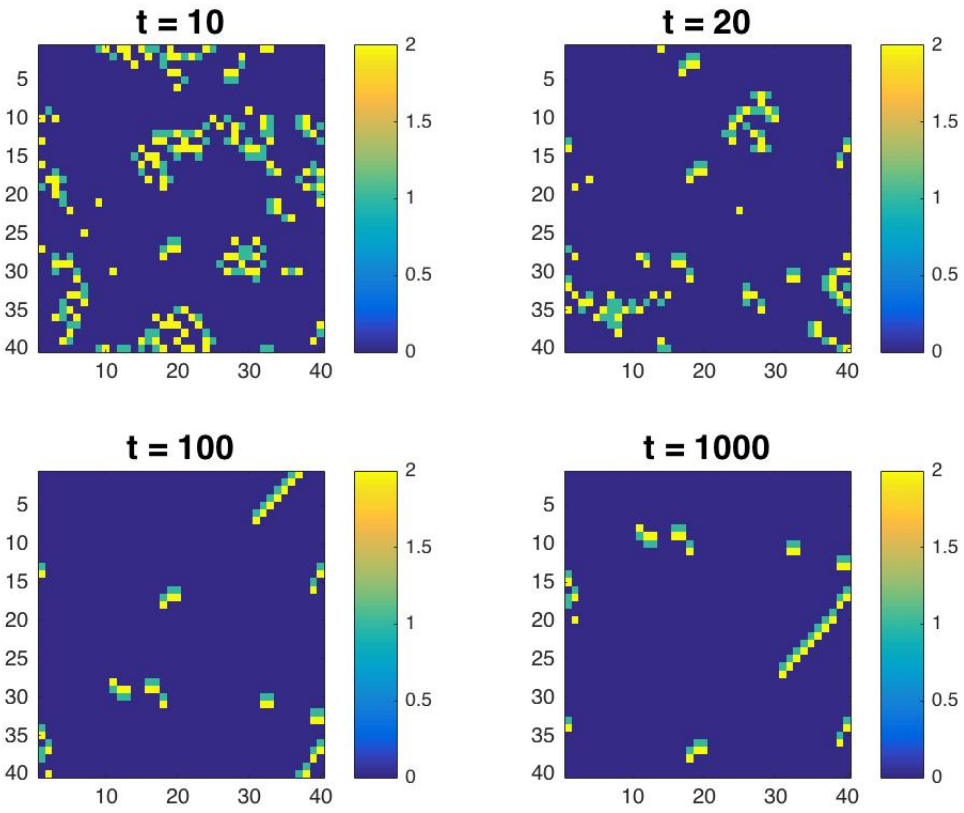


Figure 2: The grid at different timesteps

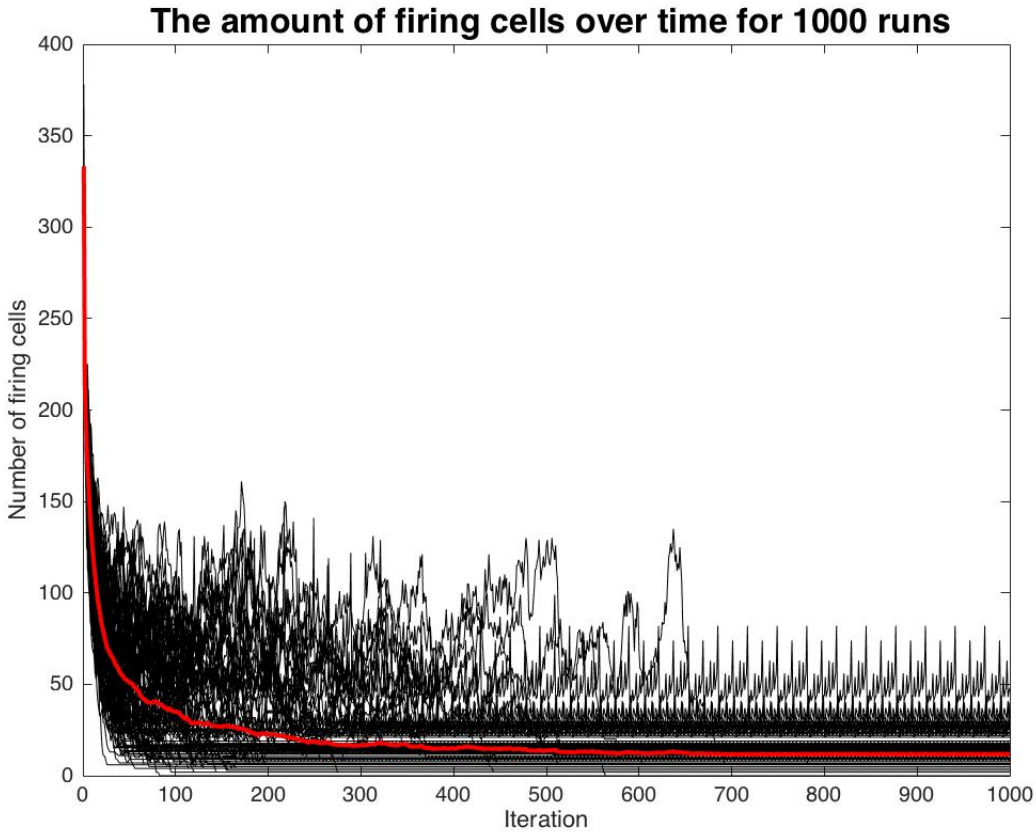


Figure 3: A graph showing the number of cells in firing state for 100 runs over 1000 timesteps. The red line is the average amount of firing cells at each timestep.

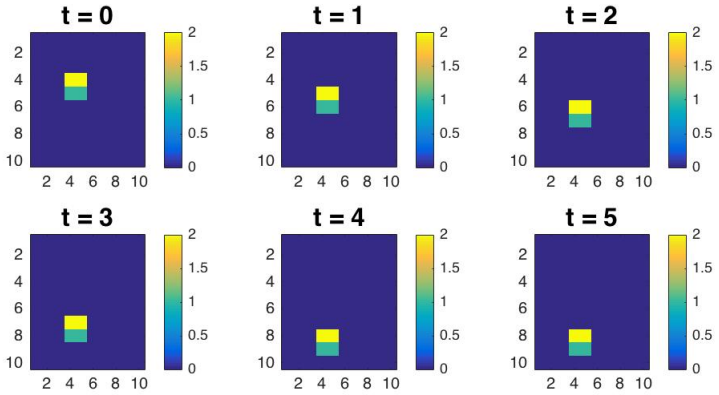
In fig.3 the system was spawned 100 times and each time allowed to run for 1000 iterations. The black lines are the sums of cells in firing state for different runs and the red line is the average number of firing cells at each iteration. From the figure it is evident that the amount of firing cells fluctuates a lot but decreases over time. After around 650 iterations, all of the systems had entered an equilibrium state in which the system had converged towards 0 firing cells, a constant amount of firing cells or (most likely) a fluctuating amount of firing cells. From fig.3 it can be noted that in none of the simulations is the amount of firing cells after iteration 650 more than 100.

1.2

1.2.1

The shape seen in fig.4 moves downwards with a rate of one step per iteration.

Figure 4: A shape that moves 1 step per timeunit and doesn't change form.



1.2.2

In the top left picture of fig.5 we have a group of cells who will move upwards with one step per time unit and will simultaneously shoot out a cell downwards as seen in the other subplots of the same figure.

1.2.3

In fig.6 the shape is moving and have returned to the same shape after 4 time units.

1.2.4

The shape seen in fig.6 is periodic and after three time steps it returns to the same shape at the same location. The whole shape has moved two units to the right and one unit upwards during these 4 iterations, thus it moves slower than one cell per iteration on average.

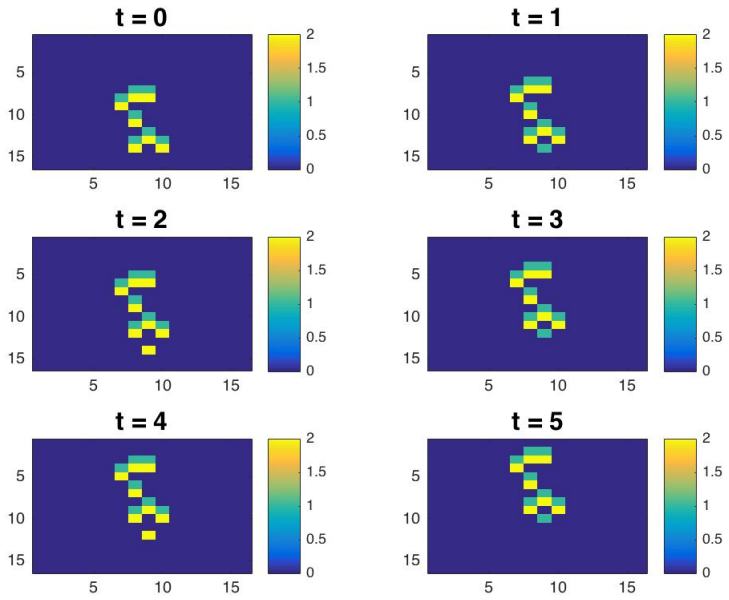


Figure 5: A group of cells that move 1 step/iteration and leave behind a cell.

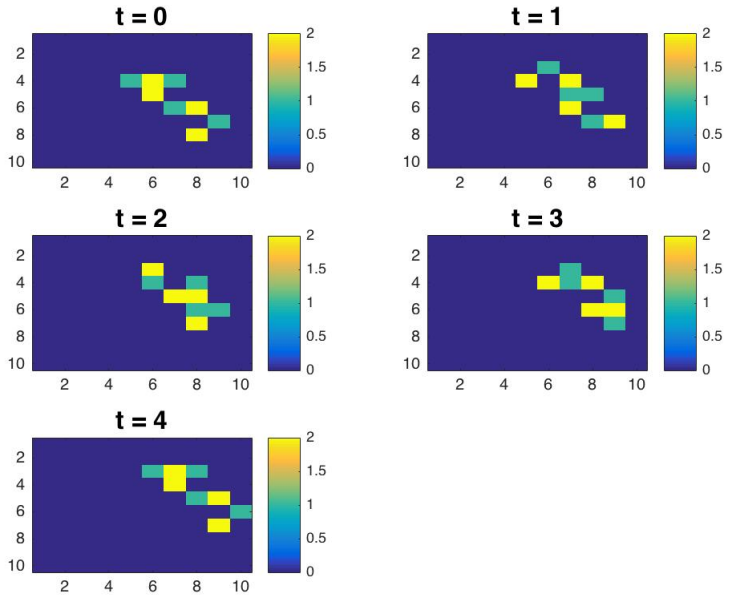


Figure 6: A shape that returns to it's form after 4 timesteps.

1.3 Custom system

In this system the cells can be in four different states {*null, prey, hunter, hunterkiller*}. For any cell, the number of neighboring null-cells is denoted N_n, the number of prey-cells N_p and so on. We have the following transitions.

$$Null \Rightarrow \begin{cases} Prey \text{ with probability } P_{N2P}(N_p) & \text{if } N_h = 0 \; and \; N_p \neq 0 \\ Hunter \text{ with probability } P_{N2H}(N_h) & \text{if } N_h \neq 0 \end{cases}$$

$$Hunter \Rightarrow \begin{cases} Hunterkiller \text{ with } P_{H2HK}(N_{hk}) & \text{if } N_{hk} \neq 0 \\ Null \text{ with } P_{H2N} = 0.1 & \text{if } N_n \geq 6 \\ Null \text{ with } P_{H2N} = 0.001 & \end{cases}$$

$$Prey \Rightarrow \begin{cases} Hunter \text{ with probability } P_{P2Hk} = \text{if } N_h \neq 0 & \\ Hunterkiller \text{ with probability } P_{N2H}(N_h) & \text{if } N_h \neq 0 \\ Null \text{ with P = 0.03 if } \; N_p > 5 \text{ and } N_h \neq 0 & \end{cases}$$

$$Hunterkiller \Rightarrow \{Prey \text{ with } P_{HK2P} = 0.3 \quad \text{if surrounded by hunterkillers and preys}$$

where

$$P_{N2P} = \begin{cases} 0.03 & \text{if } N_p = 1 \\ 0.06 & \text{if } N_p = 2 \\ 0.09 & \text{if } N_p = 3 \\ 0.12 & \text{if } N_p = 4 \\ 0.16 & \text{if } N_p = 5 \\ 0.2 & \text{if } N_p = 6 \\ 0.25 & \text{if } N_p = 7 \\ 0.30 & \text{if } N_p = 8 \end{cases} \qquad P_{N2H} = \begin{cases} 0.03 & \text{if } N_h = \{0, 1, 2\} \\ 0.02 & \text{if } N_p = \{3, 4, 5\} \\ 0.01 & \text{if } N_p = \{6, 7, 8\} \end{cases}$$

$$P_{H2HK} = \begin{cases} 0.5 & \text{if } N_{hk} = 1 \\ 0.75 & \text{if } N_{hk} = 2 \\ 0.8 & \text{if } N_{hk} = 3 \\ 0.9 & \text{if } N_{hk} = 4 \\ 0.95 & \text{if } N_{hk} = 5 \\ 0.1 & \text{if } N_{hk} = \{6, 7, 8\} \end{cases} \qquad P_{P2H} = (1 - \frac{N_n}{8}) \begin{cases} 0.3 & \text{if } N_h = 1 \\ 0.4 & \text{if } N_h = 2 \\ 0.5 & \text{if } N_h = 3 \\ 0.6 & \text{if } N_h = 4 \\ 0.7 & \text{if } N_h = 5 \\ 0.8 & \text{if } N_h = 6 \\ 0.9 & \text{if } N_h = 7 \\ 0.1 & \text{if } N_h = 8 \end{cases}$$

1.3.1 Random run

We run a simulation by initializing a system in which is 100x100 cells in size and a cell is initialized to:

$$Cell = \begin{cases} Null & \text{with } P = 0.9 \\ Hunter & \text{with } P = 0.075 \\ Prey & \text{with } P = 0.02 \\ Hunterkiller & \text{with } P = 0.005 \end{cases}$$

The initial state of the system is seen in fig.7. The colors {black, red, yellow, green} correspond {null, hunter, prey, hunterkiller}. In fig.8 the system is shown at timesteps {10, 30, 60, 100}. A video of the simulation is available online.[1] At timestep 10 it is seen that the amount of prey cells (yellow in the figure) have grown the most whereas there is only a slight increase in hunterkiller cells (green in the figure).

At timestep 30, the prey cells have formed a more robust and connected state. Since they are taking up such a large part of the grid they are more susceptible to being attacked by hunters, which is the reason to why there are some large patches of hunter cells (red in the figure). The increase in hunter cells also brings an increase in hunterkiller cells, since these prey on hunter cells. Therefore we see some slightly smaller patchs of hunterkiller cells.

In the two final subplots the system is cleared of large patches of hunter cells but we do have some spread out cells and the at each timestep some new hunter cells might spawn. More interestingly, we see that at the borders of the infrastructure that consists of prey cells, there are green cells. These green cells have thus created a defensive perimeter to protect the prey structure. against external *attacks*.

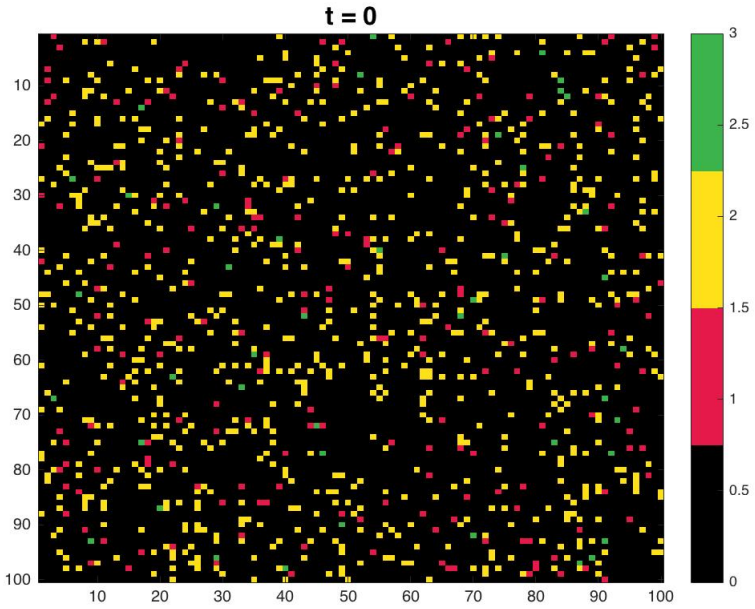


Figure 7: A grid of the cells at the systems initial state. Here the colors {black, red, yellow, green} correspond {null, hunter, prey, hunterkiller}

1.3.2 Usage of the defensive perimeter

In this section we initialize the system to have a blob of prey cells that are all inside a square of hunterkiller cells as seen in fig.9. A video can be found here.[2] The progression of the system as the simulation is being run is seen in fig.10. The growth of prey cells is contaminated to our initial defensive perimeter, with some small disfigurations. As the prey cells continue to grow/reproduce inside the square shaped structure they also become more prone to getting attacked by hunters,

[1] https://vimeo.com/263155236
[2] https://vimeo.com/263157853

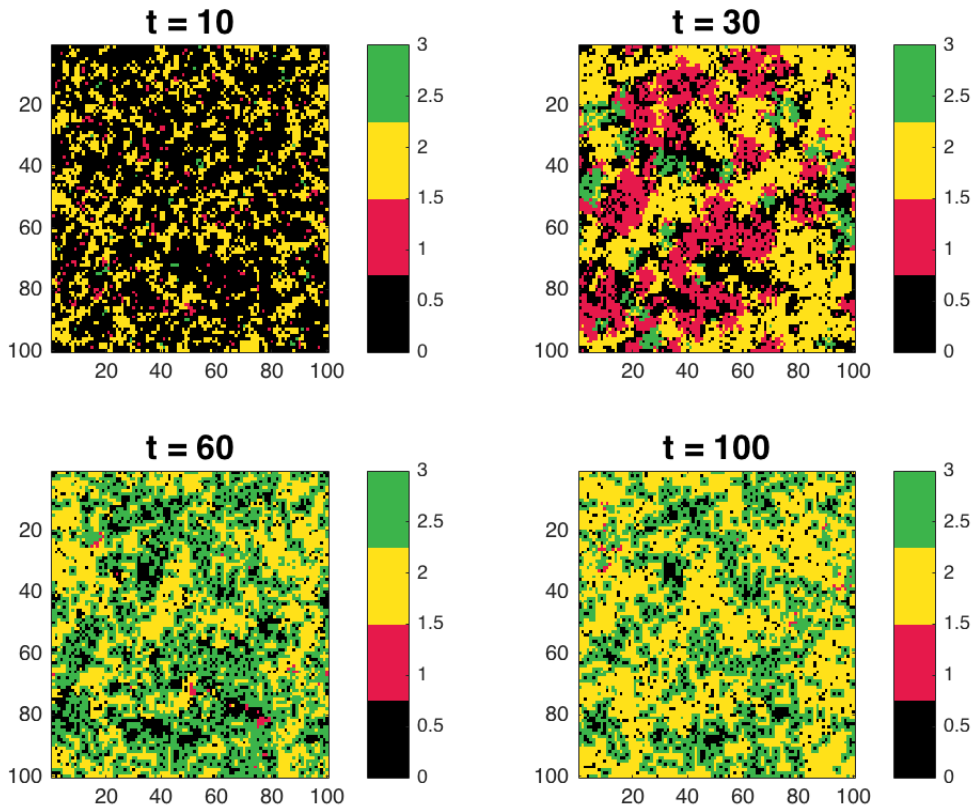


Figure 8: A grid of the cells at the systems initial state. Here the colors {black, red, yellow, green} correspond {null, hunter, prey, hunterkiller}

however the hunterkiller cells are always closeby and will defend the prey cells. This is partly seen in the subplot in fig.10 at timestep 90.

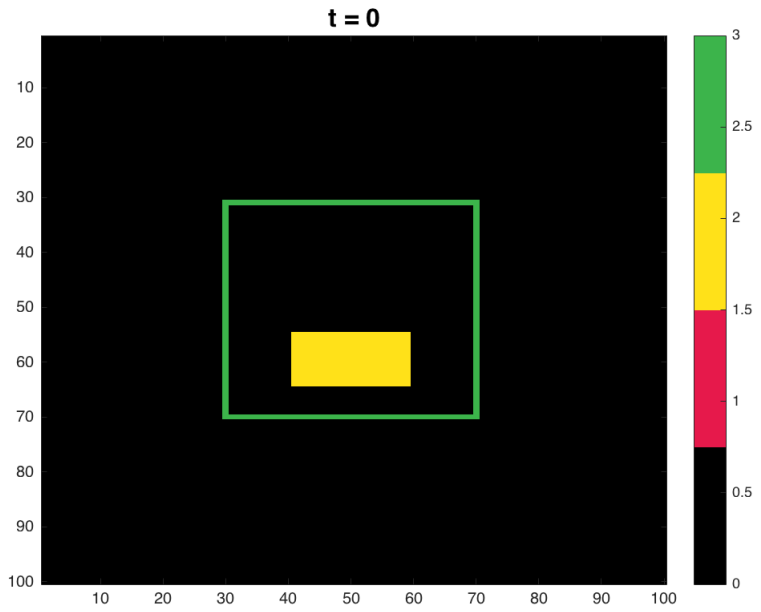


Figure 9: A bunch of prey cells inside a perimeter of hunterkiller cells

We can see that at t = 600 almost all the hunterkiller cells are at the border of our initial structure, which is now slightly disfigured. This change of shapes come from the spawning of huntercells close to the border which in turns allows the growth of hunterkiller cells. Thus the *hk* cells acts not only as a defensive parameter but also as a container of prey cells.

1.3.3 Prey or plague

Let us now see what happens if we open up the top side of our square as seen in fig.11.[3] Since there is no container of the prey cells as in the case above (see fig.10) the prey cells spread uncontrollably throughout the system and after 600 timesteps nearly the whole system is covered in prey cells as seen in fig.12.

1.3.4 Conclusions regarding our costum system

There are two ways in which we can reach something close to an equilibrium through the rules we have set. Either by letting the prey cells grow wildly and without imposing any constraints on them or by containing them within borders consisting of hunterkiller cells. In either case the hunter cells are set up to loose and will likely do so in almost all the cases. In a few percentage of the cases however, the system could end up in a state in which there are only hunter cells.

[3] https://vimeo.com/263158853

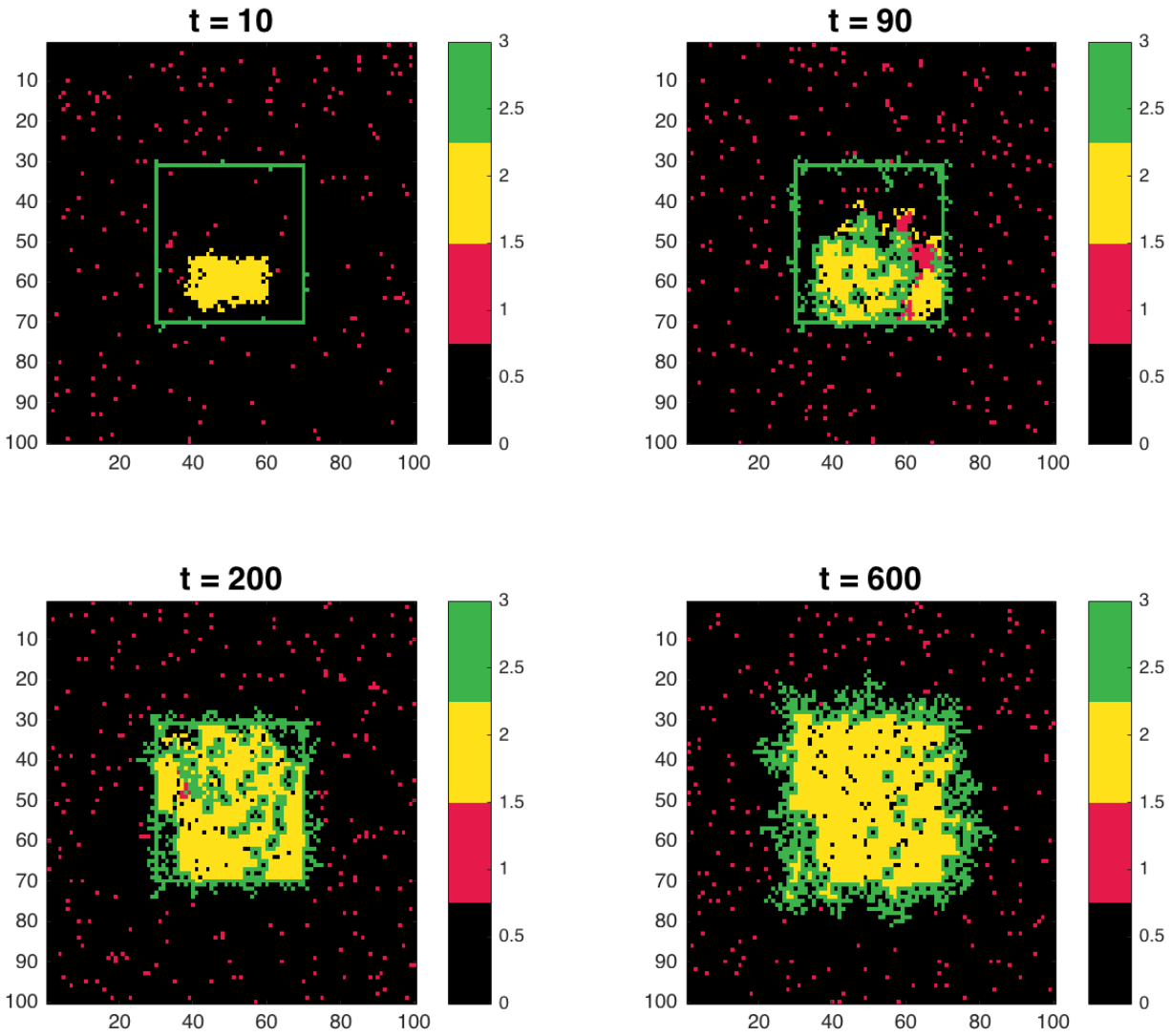


Figure 10: The development of a system in which the initial state consisted of some prey cells inside a square-shaped perimeter of hunterkiller cells.

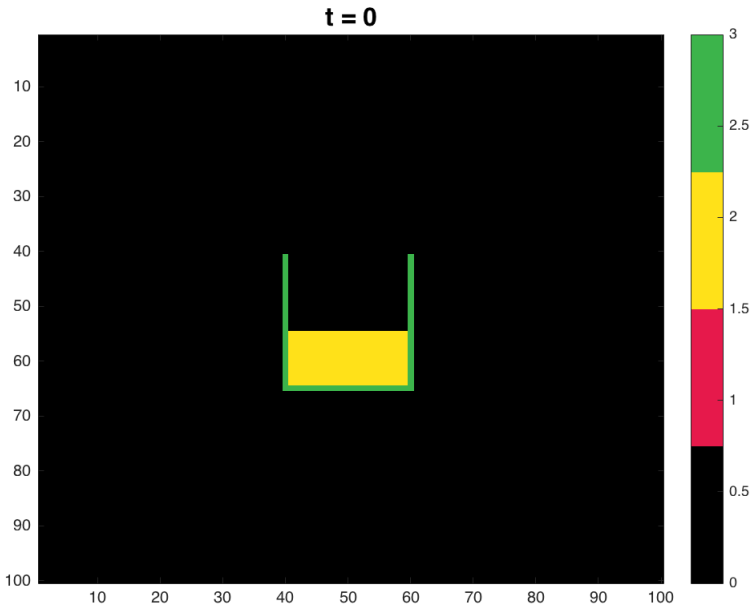


Figure 11: A bunch of prey cells inside a ”bucket” of hunterkiller cells

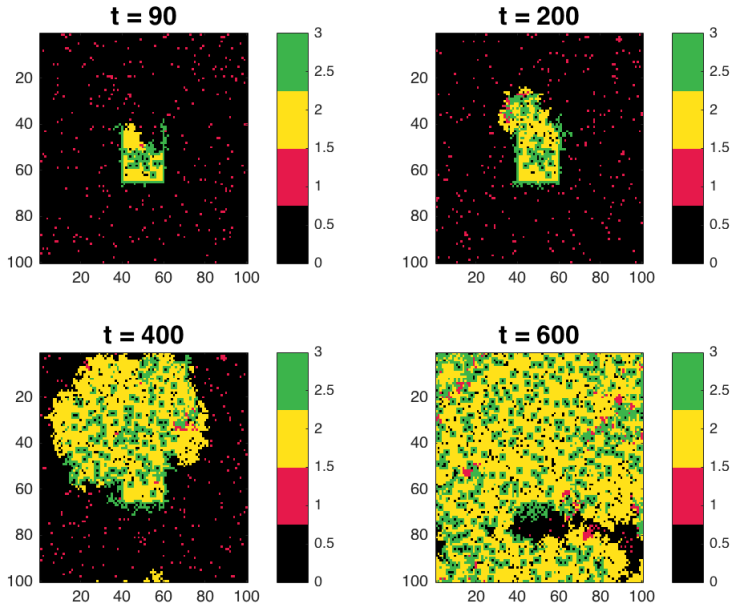


Figure 12: The development of a system in which the initial state consisted of a bunch of prey cells inside a ”bucket” of hunterkiller cells

Appendix code for part A

Main function

```
close all
clear all

N = 100;


%Initialize the matrix and randomly set the value to 0 (ready),
   1(firing)
%or 2(resting).

Pread = 0.3; %Probability of being in firing state

B_ar = zeros(N,N);
B_temp = zeros(N,N);
for i = 1:1:N
   for j = 1:1:N
      z = rand;
      if (z <Pread)
         B_ar(i,j) = 1;
      else
         B_ar(i,j) = 0;
      end
   end
end
imagesc(B_ar);
caxis([0 2]);
colorbar;
title(['\fontsize{16}t = 0']);
%At each timestep, check the neighbours and determine a new state
%for each state
r = 1;

for t = 1:1000

   for i = 1:1:N
      for j = 1:1:N
         B_temp(i,j) = newState(B_ar, i, j);
      end
   end
   B_ar = B_temp;
   %if (t == 10 || t == 20 || t == 100 || t == 1000)
   %   k = waitforbuttonpress;
   %   if k == 1
   %      error('Stopped');
   %   end
   pause(0.5)
   r = r+1
```

```
    imagesc(B_ar);
    caxis([0 2]);
    colorbar;
    title(['\fontsize{16}t = ', num2str(t)]);

    %end


end
```

newState.m Function for determining new state

```
function B=newState(A, i, j)
N = size(A,1);
if (A(i,j) == 1 || A(i,j) ==2)       %Check if it is a firing
   B = mod((A(i,j)+1), 3);           %or resting cell
else
   if (i == j && i == 1)
      Neighbours = [A(i+1,j); A(i,j+1); A(N,1); A(1,N); A(i+1,j+1);
         A(N,j+1); A(i+1, N); A(N,N)];
   elseif (i == 1)
      Neighbours = [A(i+1,j); A(i,mod(j,N)+1); A(N, j); A(i,j-1);
         A(i+1,j-1); A(N, j-1); A(i+1,mod(j,N)+1); A(N,mod(j,N)+1)];
   elseif (j == 1)
      Neighbours = [A(mod(i,N)+1,j); A(i,j+1); A(i-1, j); A(i,N);
         A(mod(i,N)+1,j+1); A(mod(i,N)+1,N); A(i-1,j+1);A(i-1,N)];
   else
      Neighbours = [A(mod(i,N)+1,j); A(i-1,j); A(i,mod(j,N)+1);
         A(i,j-1); A(mod(i,N)+1,j-1); A(i-1, j-1); A(i-1, mod(j,N)+1);
         A(mod(i,N)+1,mod(j,N)+1) ];
   end
   Num_firing = sum(Neighbours == 1);
   if (i == 10 && j == 9)
      Neighbours
   end
   if (Num_firing == 2)
      B = 1;
   else
      B = 0;
   end
end
```

Count function

```
function B=newState(A, i, j)
N = size(A,1);
if (A(i,j) == 1 || A(i,j) ==2)       %Check if it is a firing
   B = mod((A(i,j)+1), 3);           %or resting cell
else
   if (i == j && i == 1)
      Neighbours = [A(i+1,j); A(i,j+1); A(N,1); A(1,N); A(i+1,j+1);
         A(N,j+1); A(i+1, N); A(N,N)];
```

```
    elseif (i == 1)
        Neighbours = [A(i+1,j); A(i,mod(j,N)+1); A(N, j); A(i,j-1);
            A(i+1,j-1); A(N, j-1); A(i+1,mod(j,N)+1); A(N,mod(j,N)+1)];
    elseif (j == 1)
        Neighbours = [A(mod(i,N)+1,j); A(i,j+1); A(i-1, j); A(i,N);
            A(mod(i,N)+1,j+1); A(mod(i,N)+1,N); A(i-1,j+1);A(i-1,N)];
    else
        Neighbours = [A(mod(i,N)+1,j); A(i-1,j); A(i,mod(j,N)+1);
            A(i,j-1); A(mod(i,N)+1,j-1); A(i-1, j-1); A(i-1, mod(j,N)+1);
            A(mod(i,N)+1,mod(j,N)+1) ];
    end
    Num_firing = sum(Neighbours == 1);
    if (i == 10 && j == 9)
        Neighbours
    end
    if (Num_firing == 2)
        B = 1;
    else
        B = 0;
    end
end
```

Appendix code for part B

Main function

```
 % In this program the cells can be in four different states;
% null (0), hunter(1), prey(2) and hunter-killer(3).
% A null cell can turn into a null cell if it has two or more
    neighbouring
% prey-cells.
% A hunter turn into a hunter-killer if it is neighbours a
    hunter-killer.
% A prey turn into a hunter if it is surrounded by hunters and non-null
% cells (nowhere to escape). At any moment a prey can move into a
    null-cell
% unless with a certain probability p. This probability is reduced as
    the
% amount of hunter neighbours increase.

function y = Cprogram(inp)
close all
N = 100;


%Initialize the matrix and randomly set the value to 0(60%), 1(5%),
    2(30%) or 3(5%).

Pread = 0.3; %Probability of being in firing state

B_ar = zeros(N,N);
B_temp = zeros(N,N);
```

```
if strcmpi(inp, 'foundation')
   %B_ar(:,50) = 2;
   B_ar(50,:) = 2;
   B_ar(51,:) = 3;
elseif strcmpi(inp, 'seeds')
   for i = 1:1:5
      for j = 1:1:5
         B_ar(10+i,10+j) = 2;
         B_ar(40+i,40+j) = 2;
      end
   end
   B_ar(13,13) = 3;
   B_ar(43,43) = 3;

elseif strcmpi(inp, 'bucket')
   for i = 1:1:25
      B_ar(40+i,40) = 3;
      B_ar(40+i,60) = 3;
   end
   for l = 1:1:20
      B_ar(65, 40+l) = 3;
   end
   for l = 1:1:10
      for i = 1:1:19
         B_ar(54+l, 40+i) = 2;
      end

   end
elseif strcmpi(inp, 'square')
   for i = 1:1:40
      B_ar(30+i,30) = 3;
      B_ar(30+i,70) = 3;
   end
   for l = 1:1:40
      B_ar(31, 30+l) = 3;
      B_ar(70, 30+l) = 3;
   end
   for l = 1:1:10
      for i = 1:1:19
         B_ar(54+l, 40+i) = 2;
      end

   end

else
   for i = 1:1:N
      for j = 1:1:N
         z = rand;
         if (z < 0.90)
            B_ar(i,j) = 0;
```

```
            elseif (z < 0.975)
               B_ar(i,j) = 2;
            elseif (z < 0.995)
               B_ar(i,j) = 1;
            else
               B_ar(i,j) = 3;
            end
         end
      end
end
mymap = [0 0 0
   230 25 75
   255 225 25
   60 180 75]/255;

imagesc(B_ar);
colormap(mymap);
caxis([0 3]);
colorbar;
title(['\fontsize{16}t = 0']);
figure
%At each timestep, check the neighbours and determine a new state
%for each state
r = 1;
      k = waitforbuttonpress;
      if k == 1
         error('Stopped');
      end

for t = 1:1000
   t
   for i = 1:1:N
      for j = 1:1:N
         B_temp(i,j) = newStateC(B_ar, i, j);
      end
   end
   B_ar = B_temp;
%    if (t == 90 || t == 200 || t == 400 || t == 600)
%       k = waitforbuttonpress;
%       if k == 1
%          error('Stopped');
%       end
%       subplot(2,2,r);

      r = r+1
      imagesc(B_ar);
      colormap(mymap);
      caxis([0 3]);
      colorbar;
      title(['\fontsize{16}t = ', num2str(t)]);
      drawnow
```

```
      pause(0.004)
% end


end
end
```

Count.m

```
function r=count(A, i, j, state)
r = 0;
N = size(A,1);

if (j == 1)
   left = N;
   right = 2;
elseif (j == N)
   left = N-1;
   right = 1;
else
   left = j-1;
   right = j+1;
end

if (i == 1)
   up = N;
   down = 2;
elseif (i == N)
   up = N-1;
   down = 1;
else
   up = i-1;
   down = i+1;
end


if (A(up,left) == state)
   r = r+1;
end
if (A(up,j) == state)
   r = r+1;
end
if (A(up,right) == state)
   r = r+1;
end
if (A(i,left) == state)
   r = r+1;
end
if (A(i,right) == state)
   r = r+1;
end
if (A(down,left) == state)
```

```
   r = r+1;
end
if (A(down,j) == state)
   r = r+1;
end
if (A(down,right) == state)
   r = r+1;
end

end
```

Count2.m

```
function [r,t]=count2(A, i, j, state1,state2)
r = 0;
t = 0;
N = size(A,1);


if (j == 1)
   left = N;
   right = 2;
elseif (j == N)
   left = N-1;
   right = 1;
else
   left = j-1;
   right = j+1;
end

if (i == 1)
   up = N;
   down = 2;
elseif (i == N)
   up = N-1;
   down = 1;
else
   up = i-1;
   down = i+1;
end


if (A(up,left) == state1)
   r = r+1;
elseif (A(up,left) == state2)
   t = t+1;
end

if (A(up,j) == state1)
   r = r+1;
elseif (A(up,j) == state2)
   t = t+1;
```

```
end

if (A(up,right) == state1)
   r = r+1;
elseif (A(up,right) == state2)
   t = t+1;
end

if (A(i,left) == state1)
   r = r+1;
elseif (A(i,left) == state2)
   t = t+1;
end

if (A(i,right) == state1)
   r = r+1;
elseif (A(i,right) == state2)
   t = t+1;
end

if (A(down,left) == state1)
   r = r+1;
elseif (A(down,left) == state2)
   t = t+1;
end

if (A(down,j) == state1)
   r = r+1;
elseif (A(down,j) == state2)
   t = t+1;
end

if (A(down,right) == state1)
   r = r+1;
elseif (A(down,right) == state2)
   t = t+1;
end

end
```

newState.m

```
% In this program the cells can be in four different states;
% null (0), hunter(1), prey(2) and hunter-killer(3).
% A null cell can turn into a prey cell if it has a neighbouring
% prey-cells and no neighbouring hunter cells. It can also turn into a
% hunter cell with a some probability.
% A hunter turn into a hunter-killer if it is neighbours a
   hunter-killer or
% it can turn into a null cell with some probability.
% A prey turn into a hunter if it is surrounded by hunters and non-null
% cells (nowhere to escape). At any moment a prey can move into a
```

```
   null-cell
% unless with a certain probability p. This probability is reduced as
   the
% amount of hunter neighbours increase.

%Parameters:
%Probability matrix of Null => prey: [0 5 7.5 10 12.5 15 17.5 20]


function B=newState(A, i, j)
B = (A(i,j));
N = size(A,1);
z = rand;
if (A(i,j) == 0)          %If null cell
   Null_to_pray = [0 3 6 9 12 16 20 25 30]/100;
   Null_to_hunt = [3 3 3 2 2 2 1 1 1]/1000;
   [amount_of_prey, amount_of_hunter] = count2(A, i, j, 2, 1);
   if ~(amount_of_hunter) && (Null_to_pray(amount_of_prey+1) > z)
      B = 2;
   elseif (Null_to_hunt(amount_of_hunter+1) > z)
      B = 1;
   end
elseif (A(i,j) == 1)    %If hunter cell
   [amount_of_hk, amount_of_null] = count2(A, i, j, 3, 0);
   Hunt_to_hk = [0.5 0.75 0.8 0.9 0.95 1 1 1];
   if (amount_of_hk && Hunt_to_hk(amount_of_hk) > z)
      B = 3;
   elseif (z < 0.001)
      B = 0;
   elseif (amount_of_null >= 6 && z < 0.10)
      B = 0;
   end
elseif (A(i,j) == 2)    %Prey cell
   [amount_of_hunter, amount_of_null] = count2(A, i, j, 1, 0);
   amount_of_prey = count(A, i, j, 2);
   perc_of_null = amount_of_null/8;
   Prey_to_hunter = [0.3 0.4 0.5 0.6 0.7 0.8 0.9 1]*(1-perc_of_null);
   if (amount_of_hunter) && (Prey_to_hunter(amount_of_hunter)>z)
      B = 1;
   elseif (amount_of_hunter) && (0.02>z)
      B = 3;
   elseif (amount_of_prey == 8) && (z < 0.02)
      %B = 1;
   elseif (z <0.03 && amount_of_prey > 5 && amount_of_hunter == 0)
      B = 0;
   end
elseif (A(i,j) == 3)
   [amount_of_hk, amount_of_prey] = count2(A, i, j, 3, 2);
   amount_of_hunter = count(A, i, j, 1);
   if ((amount_of_hk + amount_of_prey) == 8) && z<0.3 %If completely
      surrounded by "friendly" cells
```

```
        B = 2;
    elseif (z<0.02 && ~(amount_of_hunter)) %Defence deteriorating
        %B = 2;
    end
end
```