Assignment 2: Game of Life

xxx xxxxx (xxxxx)

February 3, 2020

Contents

1 Part I

```
function ssmallgol(n,m,t)

B = round(rand(n,m)); % Rand produces random numbers between %
                      % 0 and 1. Round function rounds each %
                      % element of the matrix B to the %
                      % nearest integer; in this case to 1's %
                      % an 0's. These are the produced "cells".%

for time=1:t % Loop over the life time which is the number of %
             % iterations. %
  A = B;
  for i=1:n % Row index %
    for j=1:m % Column index %

        cola = [i-1,i,i+1]; % Creates a 3x3 matrix which are %
                            % sequential (i.e. 1,2,3) %

        cola(cola<1) = n; % This creates a periodic boundary. %
                          % If the one of the elements is 0, %
                          % the element will become n, which %
                          % is the other horizontal side of %
                          % the grid. %

        cola(cola>n) = 1; % This creates the other side of %
                          % the periodic boundary. If the %
                          % one of the elements is more than %
                          % n, the element will return to 1. %

        colb = [j-1,j,j+1]; % Increments j of a 3x3 matrix %
                            % each loop, so j continues to %
                            % be produced. %

        colb(colb<1) = m; % This creates a periodic boundary. %
                          % If the one of the elements is 0, %
```

```
                                % the element will become m, which %
                                % is the other vertical side of %
                                % the grid. %

          colb(colb>m) = 1; % This creates the other side of the %
                                % the periodic boundary. If the one %
                                % of the elements is more than m, %
                                % the element will return to 1. %

        R=sum(sum(A(cola,colb))); % This sums all the elements %
                                  % or "cells/organisms" in the %
                                  % given neighborhood. It needs %
                                  % to be summed twice because %
                                  % it is a matrix being summed, %
                                  % not a vector. %


        B(i,j) = ((R==3 || R==4) && A(i,j)) || ( R==3 && ~B(i,j));
        % This defines the rules of The Game of Life as if %
        % statements defining when the organism is "alive". If %
        % the number of organisms in the neighborhood is 3 or 4, %
        % a given organism has 2 or 3 neighbors (the 3 and 4 %
        % include the given organism itself), and "A(i,j)" means %
        % the organism is alive; therefore under these conditions, %
        % the organisms will be alive. %

        % The other condition is that if organisms have 3 %
        % neighbours and a given organism is dead, that organism %
        % becomes alive.%

        % Both these conditions, are how an organism can be alive. %
        % Any other state means the organism will be dead. %
      end
    end
    spy(B); % Spy function plots the matrix as a sparsity plot. %
    pause(0.05);
end
```

2 Part II

2.1 Population Statistics for Multiple Simulations

```
function SmallGameofLifeCode(n,m,t)
overallAvg=zeros(1,10);
subTot=zeros(5,t);

for r = 1:5

    B = round(rand(n,m));

    for time=1:t

        A = B;
        for i=1:n
            for j=1:m
                cola = [i-1,i,i+1];
                cola(cola<1) = n;
                cola(cola>n) = 1;
                colb = [j-1,j,j+1];
                colb(colb<1) = m;
                colb(colb>m) = 1;

                R=sum(sum(A(cola,colb)));

    B(i,j) = ((R== 3 || R== 4) && A(i,j)) || ( R== 3 && ~B(i,j));
            end
        end
        subTot(r,time)=sum(sum(B));
    end

    figure(1)
    spy(B);
end

    %Plot 1%
    figure(2)
```

```
plot(1:t,subTot);
title('Average Population over Time');
xlabel('Time');
ylabel('Number of live cells');

%Plot 2%
figure(3)
hold on
plot(1:t,mean(subTot))
plot(1:t,(mean(subTot)+ std(subTot)))
plot(1:t,(mean(subTot)- std(subTot)))
title('Average and Standard Deviation of Different Simulations')
xlabel('Time');
```

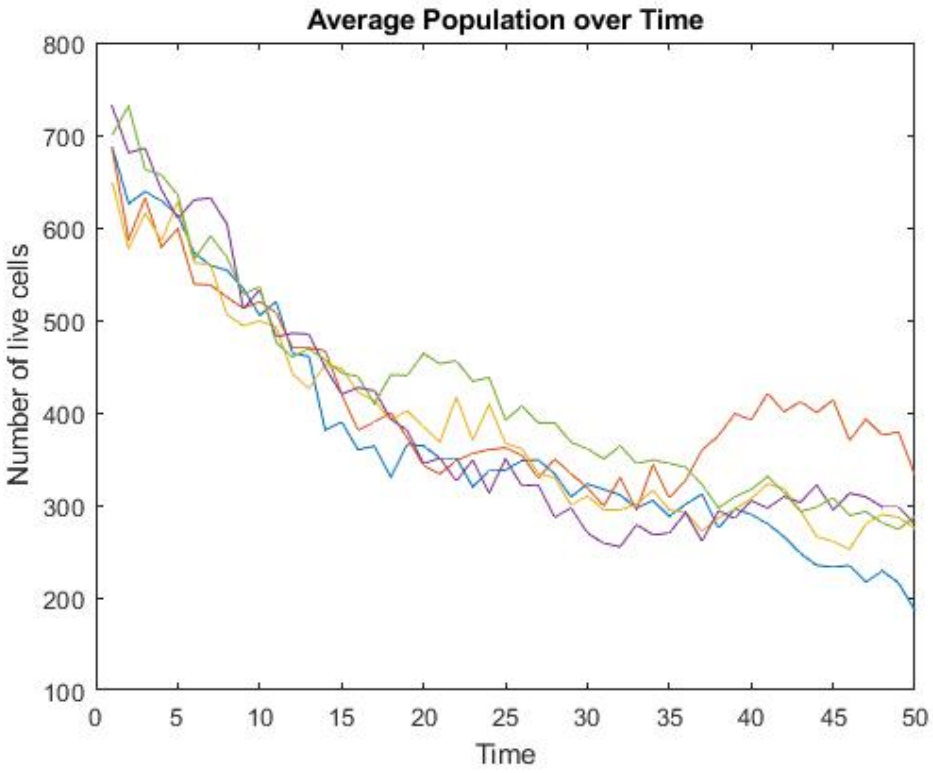

Figure 1: Average Population for each Simulation.

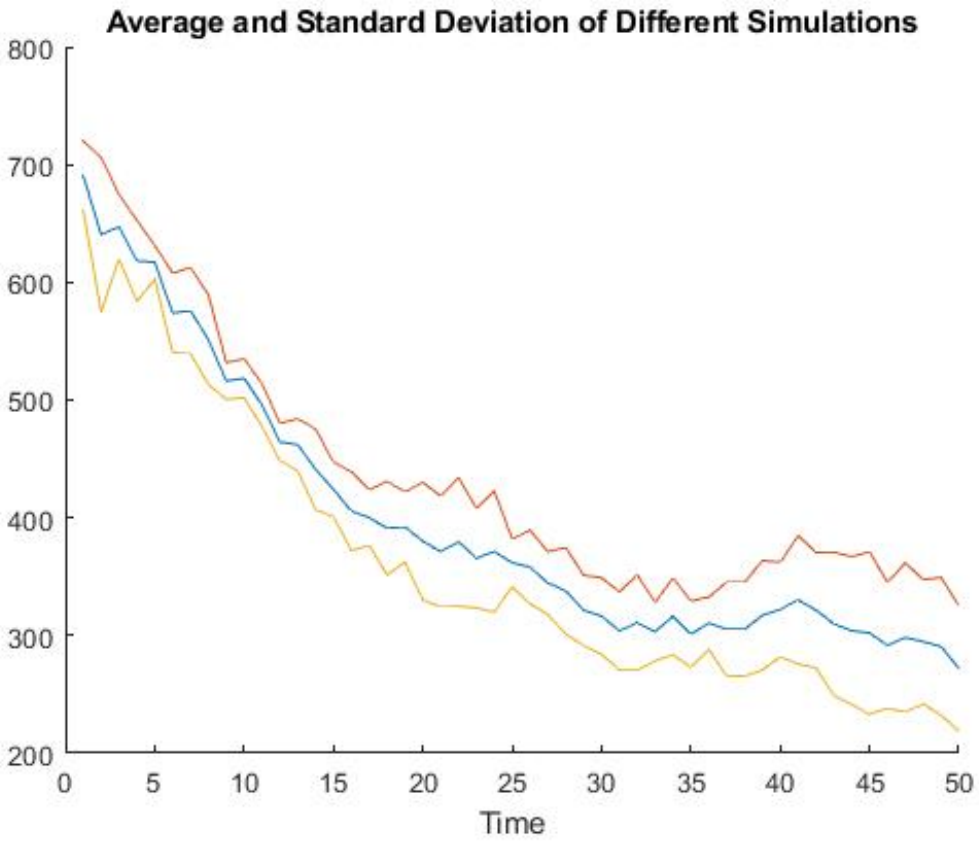

Figure 2: Population Statistics for each Simulation. Mean (Blue) and +/- Standard Deviations (Red/Yellow)

.

2.2 Varying Inputs (n,m,t)

By varying inputs of the simulations, results change. First, observe the change in results when size of simulation is increased without changing time. (>>SmallGameofLifeCode(200,200,50) in Command Window).

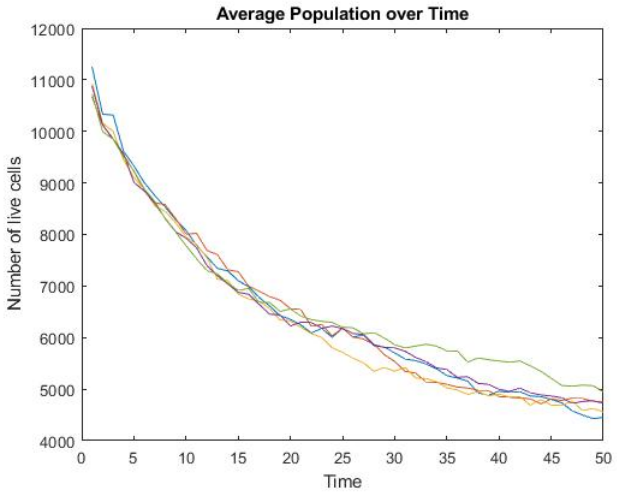

Figure 3: Average Population for each simulation.

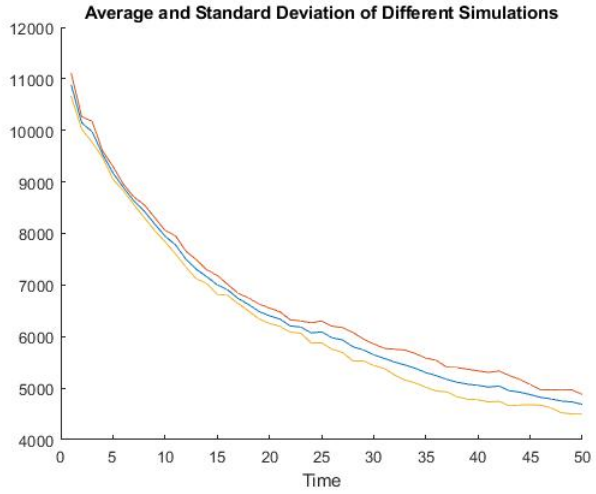

Figure 4: Population Statistics for each Simulation.

Note: The population follows a trend of a much more steady decrease. All simulations are also much more similar. The standard deviation is much smaller. This implies a bigger simulation results in more deaths, faster which more consistently result in an equilibrium. Similar to an exponential function reflected about the y-axis approaching an asymptote.

Second, observe the change in results when the time of simulation is increased without changing size. (>>SmallGameofLifeCode(50,50,500) in Command Window).

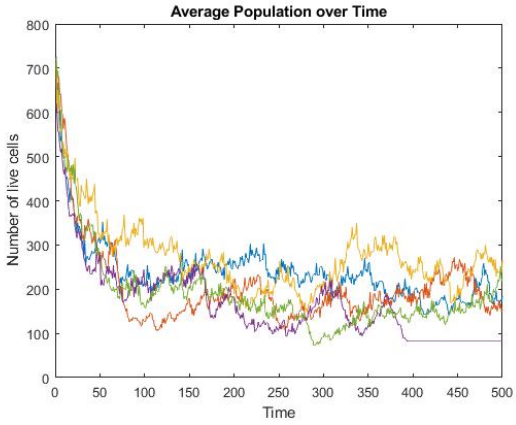

Figure 5: Average Population for each simulation

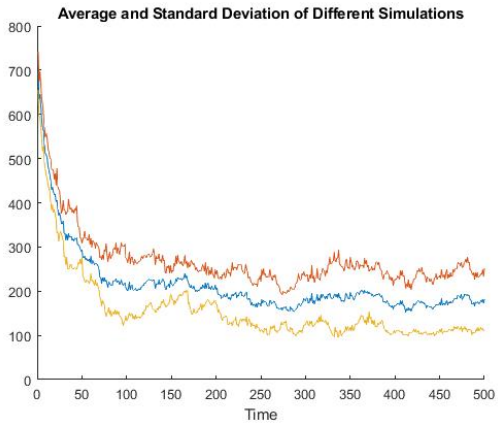

Figure 6: Population Statistics for each Simulation.

Note: Simulations differ significantly more. There is no consistent trend. Population show more randomness. As a result, standard deviation is much larger. The populations also do not die off (except the purple simulation which flat lines). This is more emblematic of a living and working ecosystem that does not die off due to overpopulation.

Finally, observe the change in results when the time and size of the simulation is increased. (>>SmallGameofLifeCode(200,200,200) in Command Window).

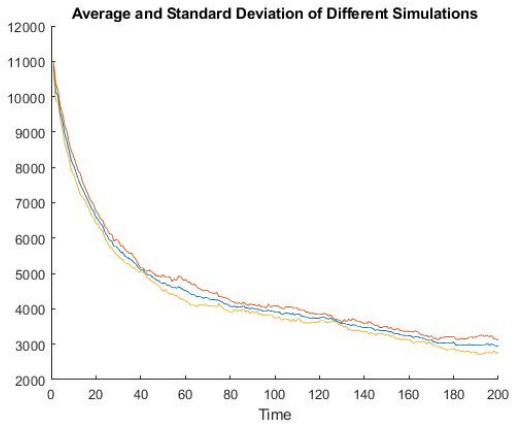

Figure 7: Average Population for each simulation

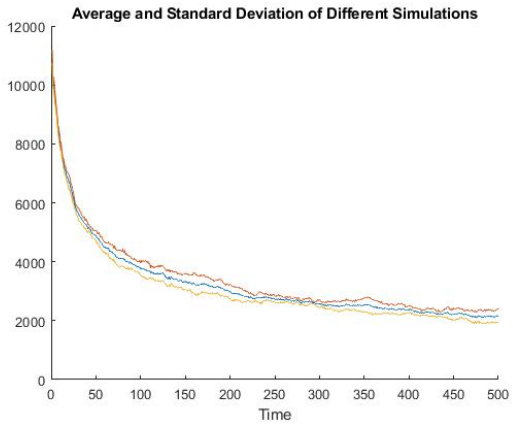

Figure 8: Population Statistics for each Simulation.

Note: Results are much more similar to when size alone was increased. The population is dies off quickly and is more consistent between simulations. However, even compared to the increased size set of simulations, the population dies off even faster. Overpopulation therefore, heavily impacts the results. Time does not impact the results very much. The population just reaches somewhat of an "equilibrium" and stays around this value (See Figure 5). Size the biggest factor which effects population.

2.3 Adding Sources and Sinks

Sources and sinks were integrated into the code by applying simple boolean if statements, where 1 is true:

```
if 1 < i && i < 10 && j > 1 && j < 30
        B(i,j) = 1; % Source %
elseif 50 > j && j > 30 && 50 > i && i > 10
        A(i,j) = 1; % Sink %
end
```

Here, in the range of the matrix 1-10 in the row index and 1 - 30 in the column index the cells cannot die. B(i,j) is always true. On the other hand, In the row index 10-50 and column index 30-50, the cells are always dead.

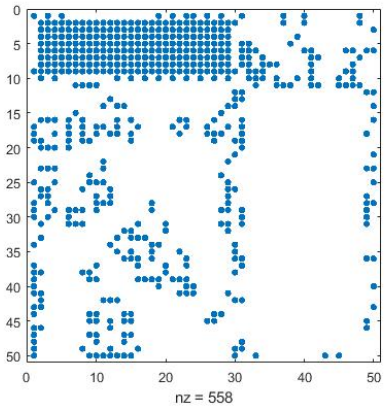

Figure 9: Sources is region filled with cells (cellular rectangle, sinks is the region with no cells (blank rectangle).

This is an external factor affecting the ecosystem. Introducing these Boolean conditions disrupts the natural state of the ecosystem. In my opinion, this sort of manifests human intervention of natural selection. Instead of a species coming to an end due to lack of resources whether it is through over/under population (first two rules of the Game of Life) or lack of independence (rule three), we stabilize this cut in population by conserving and protecting them. Similarly, we cause accelerate the extinction of species - which is manifested by the sinks in the code. Take a look at how it affects the population statistics:

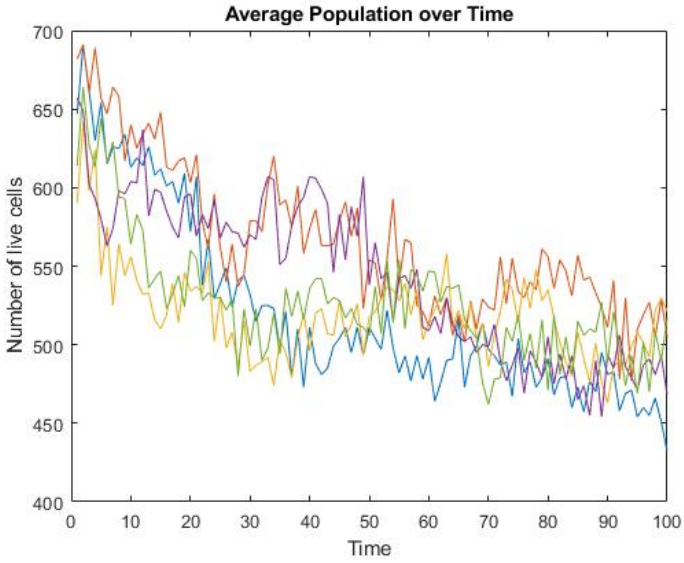

Figure 10: Population Statistics for Simulation with Sinks and Sources

Note: the population varies signifcantly more but within a relatively small range.

2.4 Removing Periodic Boundaries

Removing periodic boundaries was implemented into the Matlab code by setting the edges of the matrix to be between 1 and n (horizontal) and between 1 and m (vertical):

```
cola = [i-1,i,i+1];
cola(cola<1) = 1;
cola(cola>n) = n;
colb = [j-1,j,j+1];
colb(colb<1) = 1;
colb(colb>m) = m;
```

When the boundaries were removed population died off quickly. This is probably because the cells had no where else to go. So when cells were at the edge they were stuck and were open to many other cells entering the neighbourhood inevitably leading to overpopulation and consequentially dead cells.

2.5 Varying the Rules

When rules are changed, statistics vary significantly. For instance, when last rule is removed population immediately dies off:

```
B(i,j) = ((R== 3 || R== 4) && A(i,j));
```

When first three rules are changed the population also immediately dies off.

```
B(i,j) = ((R== N || R== (N+1) && A(i,j)) || ( R== 3 && ~B(i,j));
```

where N is an arbitrary number of neighbours. Whether the amount of cells required in a neighborhood is reduced or increased, the population still immediately dies off. If however, along with reducing the cells required in a neighborhood, the fourth rule is altered so that a dead cell with fewer neighbours becomes a live cell, the population immediately increased significantly then is maintained at this higher equilibrium. This reduces the impact of overpopulation.

```
B(i,j) = ((R== N || R== (N+1) && A(i,j)) || ( R== N && ~B(i,j));
```

3 Part III

3.1 Two-Species Game of Life

To have two species playing the Game of Life at once, the code was replicated with new variables:

```
function game(n,m,t)

B = round(rand(n,m));
D = round(rand(n,m));

for time=1:t
    A = B;
    C = D;
    for i=1:n
        for j=1:m
            cola = [i-1,i,i+1];
            cola(cola<1) = n;
            cola(cola>n) = 1;
            colb = [j-1,j,j+1];
            colb(colb<1) = m;
            colb(colb>m) = 1;

        R=sum(sum(A(cola,colb)));

B(i,j) = ((R==3 || R==4) && A(i,j)) || ( R==3 && ~B(i,j));
        end
    end
    for q=1:n
        for w=1:m
            cola = [q-1,q,q+1];
            cola(cola<1) = n;
            cola(cola>n) = 1;
            colb = [w-1,w,w+1];
            colb(colb<1) = m;
            colb(colb>m) = 1;

        S=sum(sum(C(cola,colb)));
```

```
    D(q,w) = ((S==3 || S==4) && C(q,w)) || ( S==3 && ~D(q,w));
        end
    end
    spy(B)
    hold on
    spy(D,'red');
    pause(0.05);
end
```

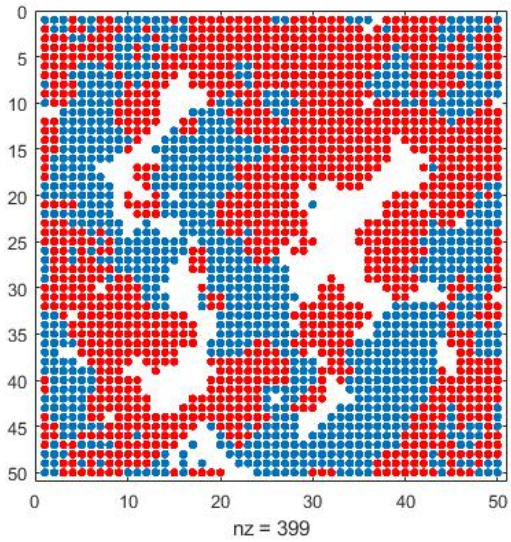

Figure 11: Sample Simulation of Game of Life running with two species.

With this code, species are able to co-exist on the same spot because the conditions do not limit the presence of the other species in its neighborhood. Almost as if it is independent code. Philosophically, this could be emblematic of a so-called "perfect utopia" with the peaceful co-existence of different species.

3.2 Aggressive Tendencies

The rules were changed so that the first species (blue) has aggressive tendencies. Assuming N is an arbitrary integer:

```
D(q,w) = ((S==3 || S==4 && R <= N ) && C(q,w) ||
          ((S==3 && R <= N) && ~D(q,w)));
```

This means whenever matrix D or species 2 has N elements of matrix B (R is the sum of the elements in its neighborhood), it will die. This denotes aggressive behaviour from species 1. The magnitude of N differently affects the population statistics. Though regardless of N, the net effect is unchanged - a population decrease. Three simulations were run with three different values of N: 3, 5, 10. The lower the N value, the more aggressive the species.

3.2.1 Very Aggressive Species

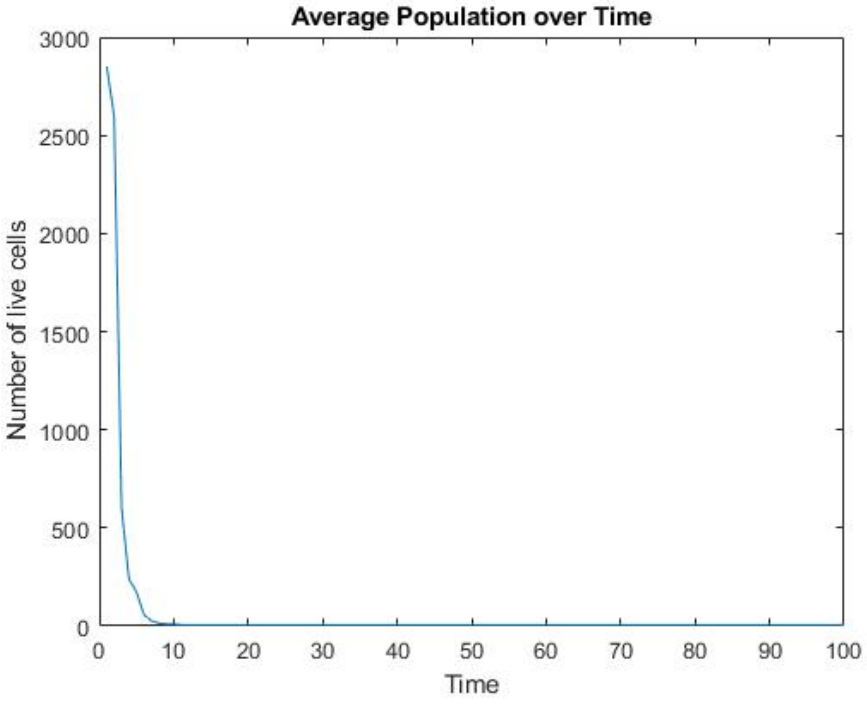

Figure 12: Population over time with N = 3

A very aggressive species which killed the other species if it had 3 neighbours. This results in an almost immediate drop in population for species 2. Within 10 seconds, the second species is almost extinct. This is obviously not a sustainable way of living. This can be seen as natural selection eliminating a species too weak to sustain itself within the predatory ecosystem.

3.2.2 Aggressive Species

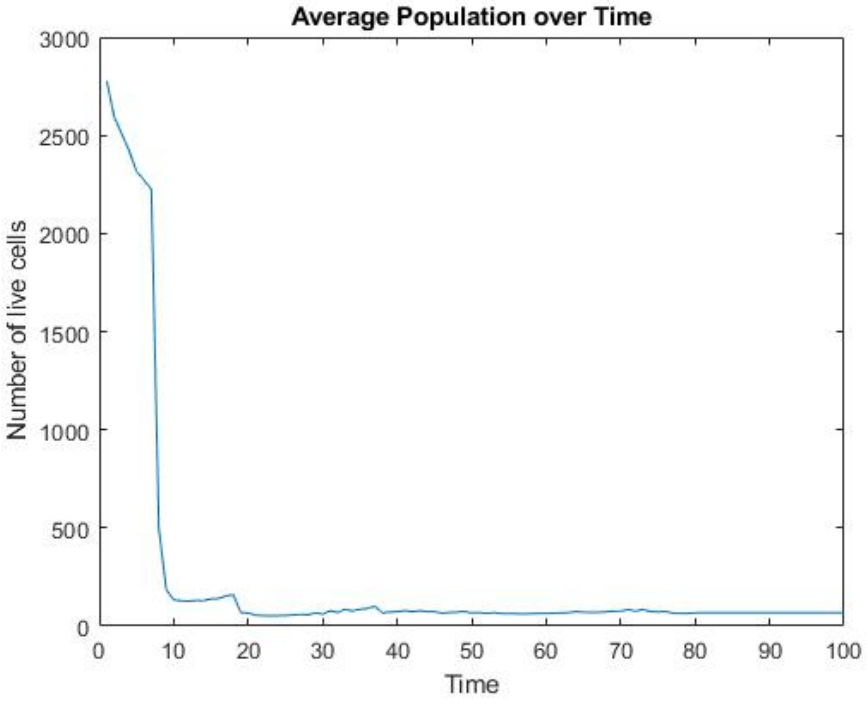

Figure 13: Population over time with N = 5

Similar to the very aggressive species, the population dies off extremely quickly. However, instead of becoming extinct it has approached a small equilibrium. Therefore, the species is weak but able to sustain life in the predatory equilibrium.

3.2.3 Mild Aggressive Species

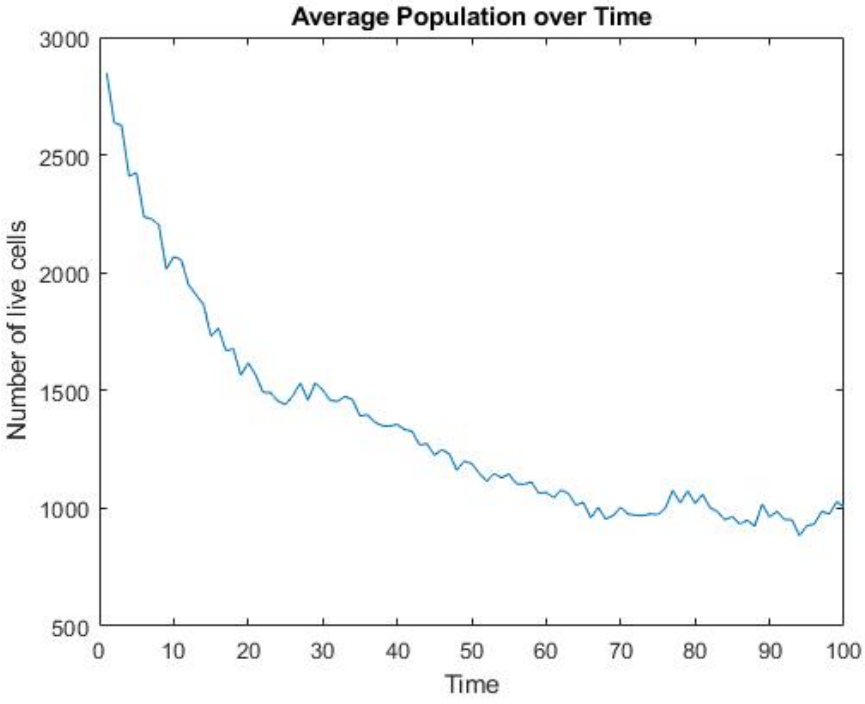

Figure 14: Population over time with N = 10

The mild aggressive species emulates a much more realistic simulation of a predatory ecosystem. Here, within the first, approximately, 20 seconds, there is a relatively quick drop in population; after this, time-range the rate at which the population decreases quickly. The equilibrium is at a reasonable population. The prey species is able to continue to sustain life.

3.3 Asymmetric Aggression

The net effect of all variations of N is a decrease in population. This asymmetric aggression (one species is consistently dominant) shows the macroscale concept of natural selection extremely quickly within the simulation. The aggressive species inevitably kills the other species at a very high rate. Once there are a fewer number of that species, it is more difficult for the predator to find another prey to kill.Hence, the equilibrium is reached. With

less aggressive species, that equilibrium is higher (*See 3.2.3 Mild Aggressive Species*).