ELEC 1908

Simulation and Analysis of Conway's Game of Life

Authors:
Kliti Bala
Denis Shleifman
David Kemdirim

Professor: Tom Smy

Date Submitted: February 28 2016

1 Abstract

The purpose of this assignment is to get familiar with Conway's *Game of Life*. The *Game of Life* will be explained and introduced in detail. Then, a sample *Game Of Life* program will be analyzed and described. After this is completed, various statistical information regarding the *Game of Life* will be gathered using MATLAB simulations, exploring the effect of various game attributes on the world population. The effect of variables such as world size, timespan, sources and sinks, as well as periodic boundaries on world population will be analyzed and conclusions will be drawn from this data. After this is completed, the rules of the *Game of Life* will be altered to include multiple species, with a variety of attributes. The resulting Game of Life will be analyzed in detail, and conclusions will be drawn. Finally, an optimized *Game of Life* program created with the aim of increasing algorithmic efficiency will be created. The runtime of this program will be compared to the runtime of a conventional *Game of Life* algorithm programmed in both C and MATLAB, and a final conclusion on the effectiveness of the new algorithm will be drawn.

2 Introduction to Conway's *Game of Life*

Conway's *Game of Life* is a popular simulation extensively studied in computer science and mathematics. The game consists of an orthogonal grid of square cells, which can have be in two states; alive or dead. The state of each cell is dependent on a set of rules/conditions.

1. Any live cell with fewer than two live neighbours dies, as if caused by underpopulation.
2. Any live cell with more than three live neighbours dies, as if by overcrowding.
3. Any live cell with two or three live neighbours lives on to the next generation.
4. Any dead cell with exactly three live neighbours becomes a live cell.

A program that simulates the *Game of Life* based on these 4 rules can then be constructed, as seen in **Figure 1.**

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

    B = round(rand(n,m)); %generate nxm world matrix

    for time=1:t %for time start at 1 until equal to t
        A = B; %Set holder matrix A equal to world matrix B

        for i=1:n %Iterate through each row
            for j=1:m %iterate through each column
                cola = [i-1,i,i+1]; %Generate 3 col row composed of i-1,i,i+1
                cola(cola<1) = n; %if less element in colb than 1, set equal to n
                cola(cola>n) = 1; %if element in colb greater than n, set equal to 1

                colb = [j-1,j,j+1];%Generate 3 col row composed of [j-1,j,j+1]
                colb(colb<1) = m;%if element in colb less than 1, set equal to n
                colb(colb>m) = 1;%if element in colb greater than n, set equal to 1

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

                % Cell A(i,j) is live if R = 3 or 4 and A(i,j) =1 or if r =3 and entryB(i,j) is dead
                B(i,j) = ((R==3 || R==4) && A(i,j)) || ( R==3 && ~B(i,j));
            end
        end

        spy(B);
        pause(0.05);

    end

end
```

Figure 1: Source code of sample *Game of Life* program written in MATLAB

The code seen in **Figure 1** works by first spawning a world with a random configuration of live and dead cells. After this is done, the program then checks every cell in the world, checking how many neighbouring live cells the cell has, and the state of each cell. Based on this information and the rules of the *Game of Life*, it sets every cell in the world to alive or dead. After this is completed, one generation of the *Game of Life* has been determined, and the program then goes to determine the state of the next generation, until the last generation of the world, specified by the user, has been reached.

3 Statistical Analysis of *Game of Life* Program

3.1 General Population Trends

3.1.1 Population vs Time

The first portion of statistical and data analysis of the *Game of Life* focused on modeling population as a function of time. The first simulation performed was done on a 100x100 world matrix, run for 2000 generations,as seen in **Figure 2**.

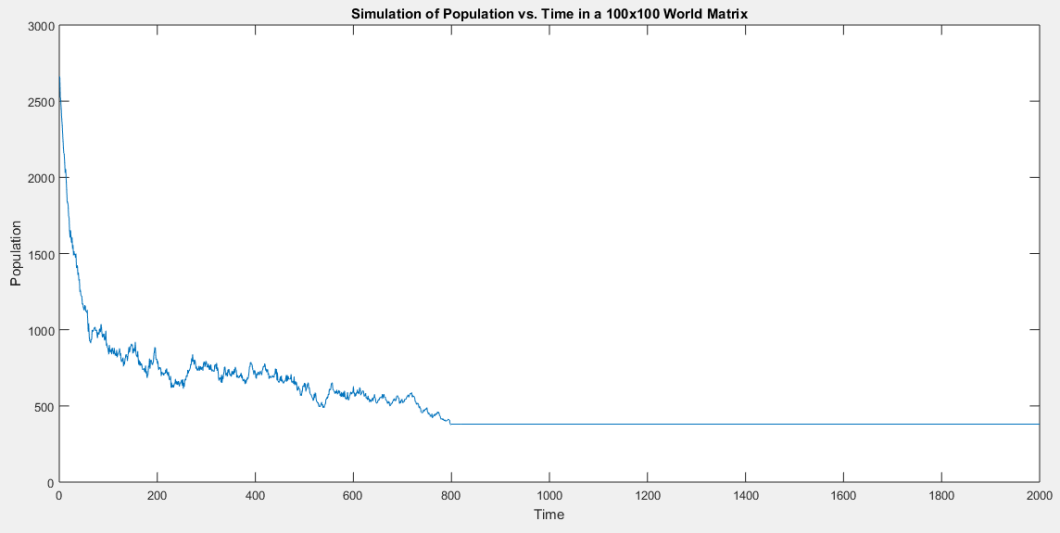

Figure 2: Population vs. Time in a 100x100 World Matrix

The observations in **Figure 2** give rise to a hypothesis of how population progresses in the *Game of Life*. There appear to be a number of stages of progression.

Stage 1: The initial generations of a world appear to be highly unstable, as indicated by the plummet in population in the initial generations. **Stage 1** can be seen in **Figure 2** in the time interval $0 < Time < 50$. As will be mentioned later, this stage does not occur only in the beginning

Stage 2: After this initial plummet, the world then appears to transition to a more stable state. The population then cycles between two states; an era of population increases and decreases. Populations appear to decrease down to a stable configuration with enough space to grow, and then increase in population to the point that they become unstable and a large number of cells die off. While in some intervals this cyclicization in population leads to a consistent increase, the general trend appears to be leading to a consistent drop in population. Despite this trend of population decay, the cycles of increase and decrease still occur, but appear to damp over time in amplitude. **Stage 2** can be seen in **Figure 2** in the time interval $50 < Time < 800$

Stage 3: The state of population in a world appears to then decay into a final state of equilibrium. In equilibrium, the world is populated only by configurations of cells that oscillate between different geometric configurations, but nevertheless are stable and do not die out. These configurations of cells are called oscillators. At the point of equilibrium, the population remains flat, and the world only observes these ocillators switching between their various states. **Stage 2** can be seen in **Figure 2** in the time interval $800 < Time < 2000$

After observing these trends in the first *Game of Life* simulation their valitidy as general trends was tested, by running multiple *Game of Life* simulations and observing if these trends hold. Five different simulations, seen in **Figure 3** were performed.

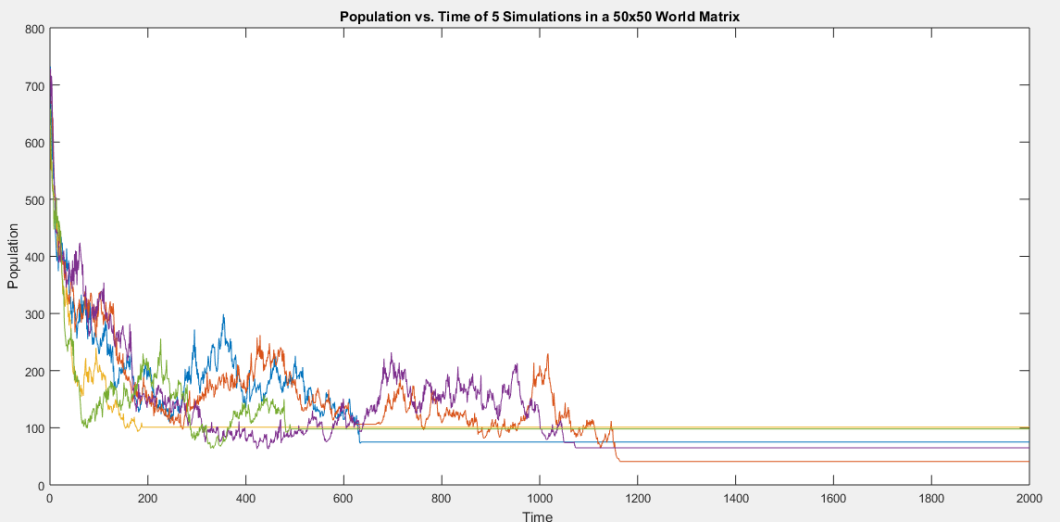

Figure 3: Five different simulations of Population vs. Time in a 100x100 World Matrix

As can be seen in **Figure 3**, these trends do hold. In every simulation, there was the observed trend of an initial population plummet, state of stability coupled with oscillations, a damping of these oscillations, and a progression and transition to a final and unchanging state of population equilibrium. The validity of these trends is made even more apparent when graphing the average population as time progresses. When this is done, there is a clear and unmistakable downwards trend toward a state of unchanging population, filled with only oscillators, seen in **Figure 4**. Given these extra simulations, it appears the aforementioned trends and progressions in the *Game of Life* do and can generally, but not in all cases describe population progression in the *Game of Life*. This is made apparent in **Figure 5** in **Section 3.2** where population progression does not transition smoothly between these stages. In **Figure 5**, while all the three aforementioned stages were observed, some of the simulations in **Figure 5** transitioned between **Stage 1** and **Stage 2**, plummeting, reaching a period of stability,plummeting again, and regressing back into **Stage 2**, and then after interchanging between these stages, going into **Stage 3** equilibrium. As a result, it should be mentioned that while these trends hold, they do not progress linearly from one to the other and populations can interchange between **Stage 1** and **2**.

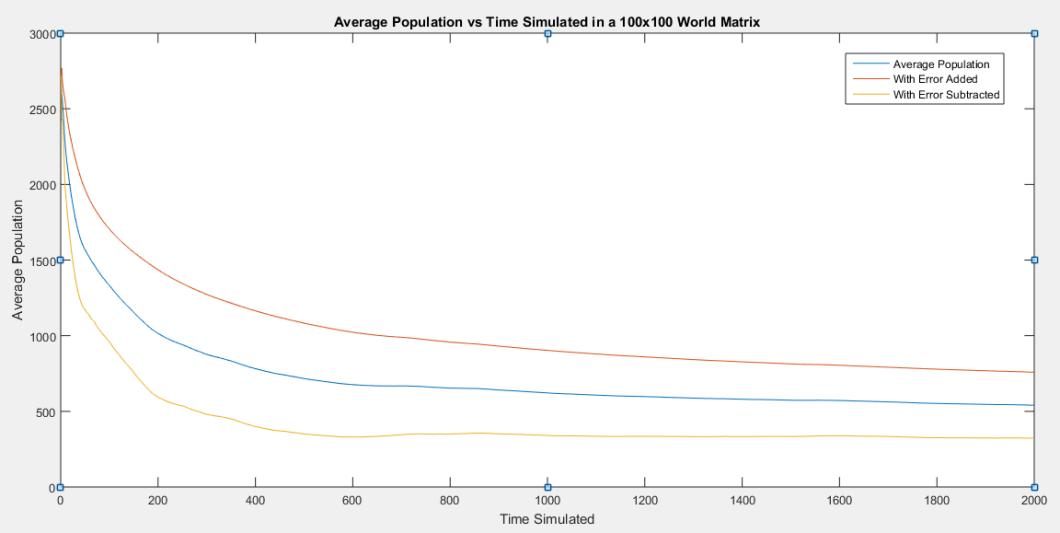

Figure 4: The average population of a world with respect to time passed with standard deviations

3.1.2 Population vs World Size

The effect of the initial world size can also have an effect on the population progression. As seen in **Figure 5**, the population progression varied with a change in world size, although not by much. The only major change observed was the overall population which became larger with an increase in world size. With an increase in world size also came larger oscillations in **Stage 2**, due to the extra growing room cells had. Otherwise, the general population trends did not change by much.

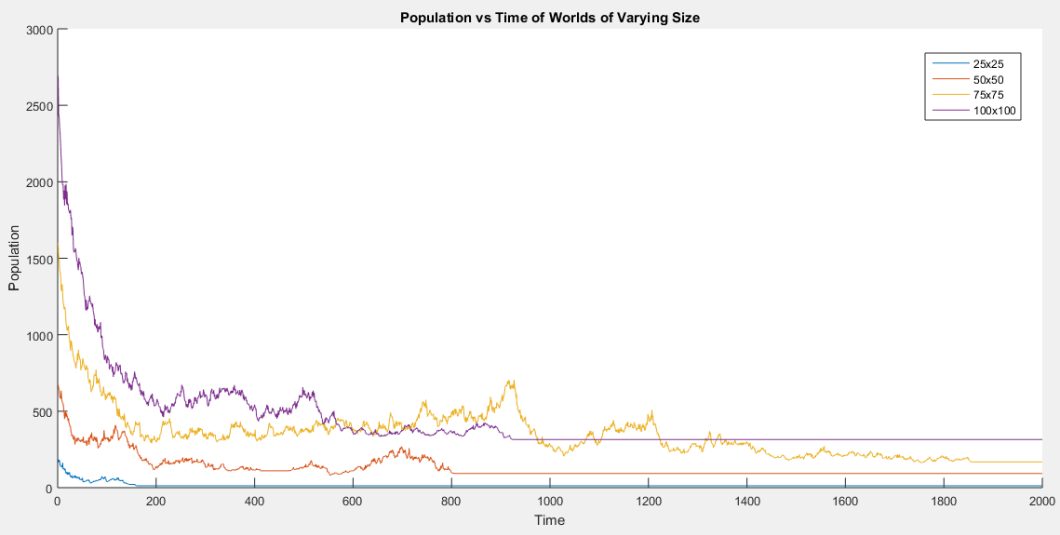

Figure 5: Population vs Time of Worlds of Various *nxn* size, as seen in the legend

3.2 Effect of Sources and Sinks on Population

The *Game of Life* is a highly malleable simulation and the rules can be changed around in many different ways. One alteration to the original rules of the game is to include sources and sinks. Sources and sinks are special kinds of cells which do not change state. A source cell is always alive and and a sink cell is always dead. With these additional types of cells in the game, the evolution of population vs time changes.

As can be seen in **Figure 6**, the addition of sources and sinks drastically alters the population as a function of time depending on what ratio of sources and sinks are used. It should be mentioned that all the programs used the same initial world matrix, only adding sources and sinks after receiving the initial live cells.

With no sources or sinks present in the system, the population vs time graph progresses like a typical *Game of Life*, in the aforementioned stages, finally reaching a state of unchanging equilibrium.

With the addition of 199 sink cells and one source cell, the Population vs Time graph clearly changes. With a large amount of permanently dead cells, the population plummets downwards at a far faster rate than a normal simulation would. Furthermore, the period of a stable trend downwards in **Stage 2** becomes a linear trend downwards towards a state of equilibrium, compared to the regular simulation,which has a much flatter slope downwards. Furthermore, it is apparent that the curve downwards is devoid of any **Stage 2**, rather simply progressing downwards in population with no period of stability and oscillations characterized by **Stage 2**. This seems to indicate that sink cells prevent populations from going down to a stable configuration and then growing in population to an unstable one as seen in a regular **Stage 2** of the *Game of Life*, as any rise in population would be prevented by the presence of so many sink cells. With only sink cells

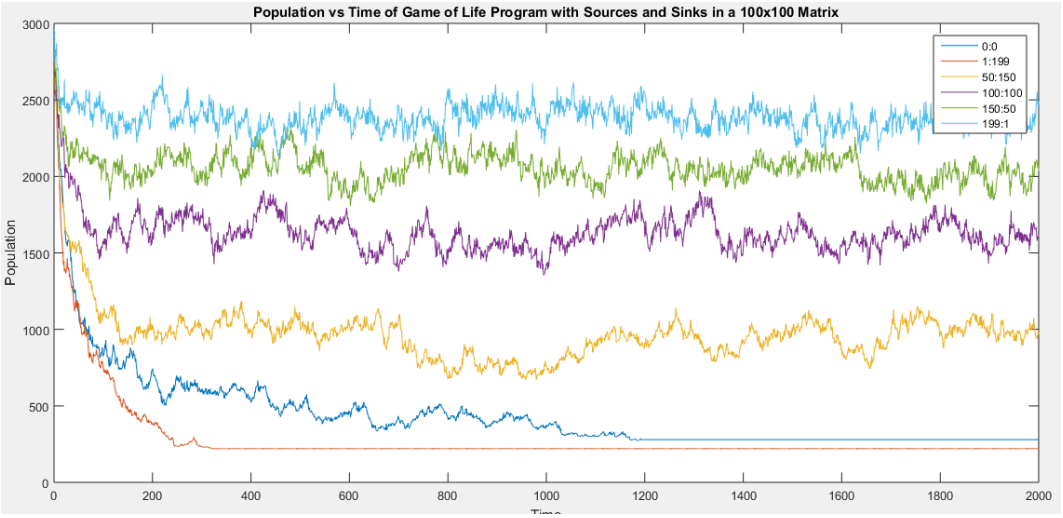

Figure 6: Simulations of the **Game of Life** with varying source to sink ratios, as seen in the legend

the time required to reach equilibrium also becomes much shorter compared to a regular simulation, hitting equilibrium approximately 700 generations.

With the addition of only 50 source cells compared to 150 sink cells however, the plummet in population almost completely disappears, which is a surprising result, as a high sink to source ratio should still lead to a population curve observed in the 1:199 source sink ratio. Rather, the population goes down to a stable **Stage 2**, although this stable population level is higher than the regular simulation. When **Stage 2** is reached, the population then begins its regular period of oscillations in population, although they are much longer in period. This is likely due to a control on the rate of population increase and decrease by the sink cells. Finally, rather than reaching equilibrium, the population simply remains in **Stage 2**. It could be that the world reaches equilibrium far later than a regular *Game of Life* simulation due to the sources and sinks, but it seems more likely that equilibrium never actually occurs unless you run the world an extremely large amount of generations, due to the fact that any contraction in population is prevented from being too large by sources, meaning the population will not reach a state where it is unable to increase, unless it is in a very specific geometric configuration where live cells have no nearby sources to help them grow.

For 100:100 and 150:50 and 199:1 source sink ratios, the same trend observed in the 50:150 source sink ratio occurs. The only difference observed however, is that the initial population plummet of **Stage 1** is much lower and the stable population of **Stage 2** is much higher. Also, the oscillations increase in frequency when there are more sources than sinks, further giving credence to the hypothesis that sinks decrease the rate of increase and decrease.

In conclusion, sources and sinks affect the way that population progresses over time. A program

with only sinks has a much larger plummet in **Stage 1**, and has almost no period of stability in **Stage 2**, also reaching **Stage 3** equilibrium far faster than a regular *Game of Life*. The addition of sinks prevents a large **Stage 1** population plummet and maintains a long **Stage 2** with much lower frequency oscillations and prevents equilibrium, even with a high sink to source ratio.

3.3 Effect of Removing Periodic Boundaries of World Matrix on Population Trends

The *Game of Life* is simulated in an nxm matrix representing the world. However, one can change this so that the world is a quasi-infinite matrix, which expands in size whenever there are cells on the edge, so that there is technically no boundary in the world. The population vs time graph was once again graphed, as seen in **Figure 7**. It should be mentioned that both simulations used the same initial world matrix.

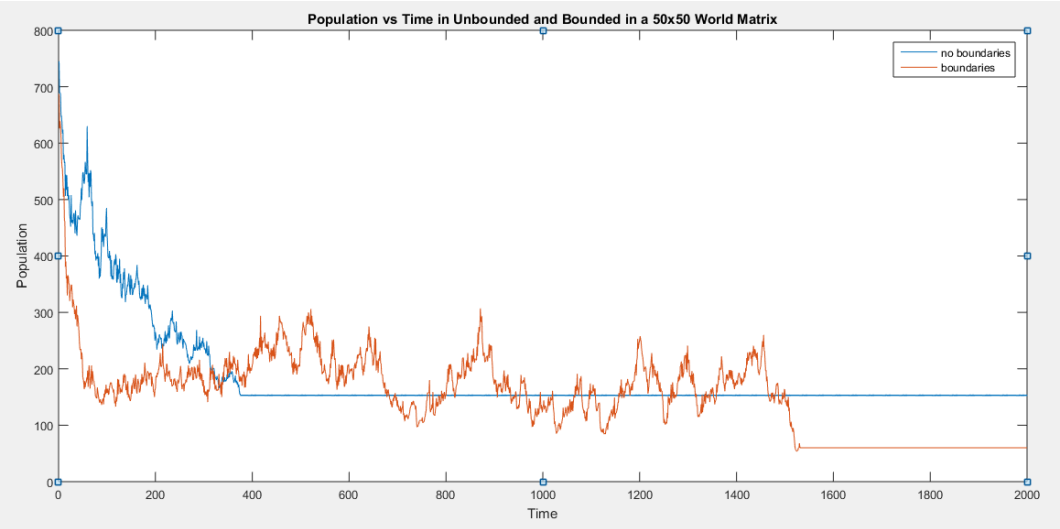

Figure 7: Population vs Time of a bounded and unbounded world

As can be seen in **Figure 7**, there is a clear and noticeable difference between a bounded and unbounded world matrix. Firstly, the plummet in **Stage 1** is much lower in the unbounded world compared to the bounded world. This can be explained by the fact that while there is still a large reduction in population due to the deaths of unstable cells, this is mitigated by the fact that the world is unbounded and cells at the edge can continue to grow, therefore making the net population loss smaller,thereby lowering the plummet in **Stage 1**. The population stability of **Stage 2** is not observed in the unbounded simulation, compared to the bounded simulation where there is clearly a long period of **Stage 2** population stability. As can be seen in **Figure 7**, the population plummets down to a state of equilibrium almost immediately after **Stage 1**. While the population graph prior to equilibrium is not a steep drop, it is a consistent drop in population with no flat stable period as

observed in the bounded graph, where population remains in a relatively stable period of oscillation. This is quite unexpected, because even without boundaries, there should be some period of relative population stability like in **Stage 2**. This could be because as population increases outwards unchecked by the boundaries, population concentration decreases and the periods of stability and oscillation can't occur due to low neighbour count, and as a result populations cant even reach **Stage 2** but transition to equilibrium immediately. This hypothesis becomes even more plausible when graphing population count and live cell concentration as a function of time in **Figure 8**.

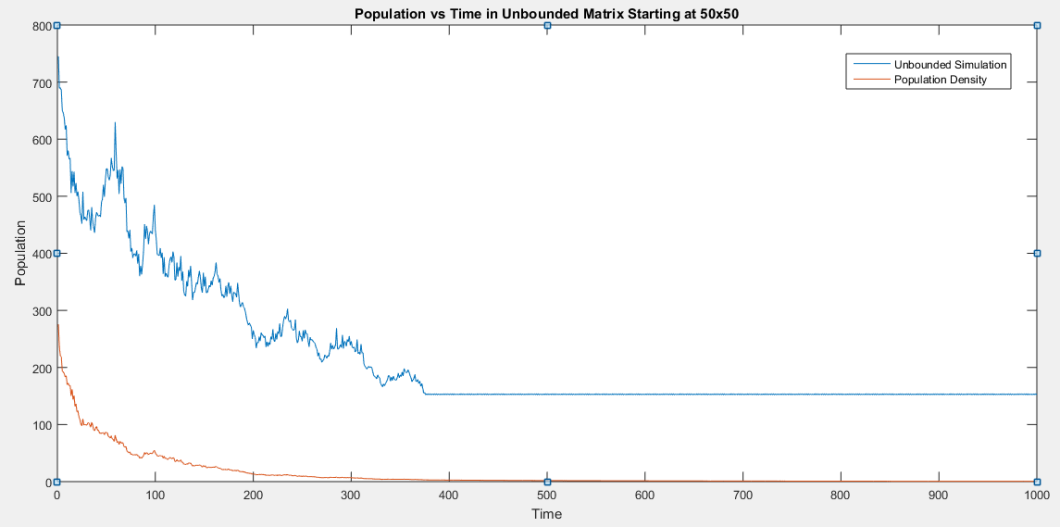

Figure 8: Simulation of an Unbounded 50x50 World Matrix as well as Population Density adjusted in size to be made visible

As can be seen, most notable in the time interval $50 < Time < 100$ even as the population spikes up, net concentration still goes down. This leads to an increase in population temporarily due to the increase of population at the world edges but in the end reduces population density too much, preventing a stable **Stage 2**, but leaving a large population of oscillators behind. Due to the lack of boundaries, populations can expand indefinitely even as the unstable cells are being killed off, resulting in a much larger population of oscillators, and as a result, a larger equilibrium population.

3.4 Effect of Changing Game Rules

The conventional *Game of Life* goes by two simple rules for determining if a cell lives or dies. If the cell is alive and it has 3 or 4 neighbours it lives on to the next generation. If it is dead and has 3 neighbours, it becomes live. Any other state and the cell dies. However, there can be variations to these rules, which will change how population progresses over time.

3.4.1 Changing the Rules for Live Cells

Changing how many cells an already live cell needs in order to remain alive for the next generation gives some interesting results. Eight different simulations were run, each with different rules for how many live neighbours a live cell needed to remain alive.

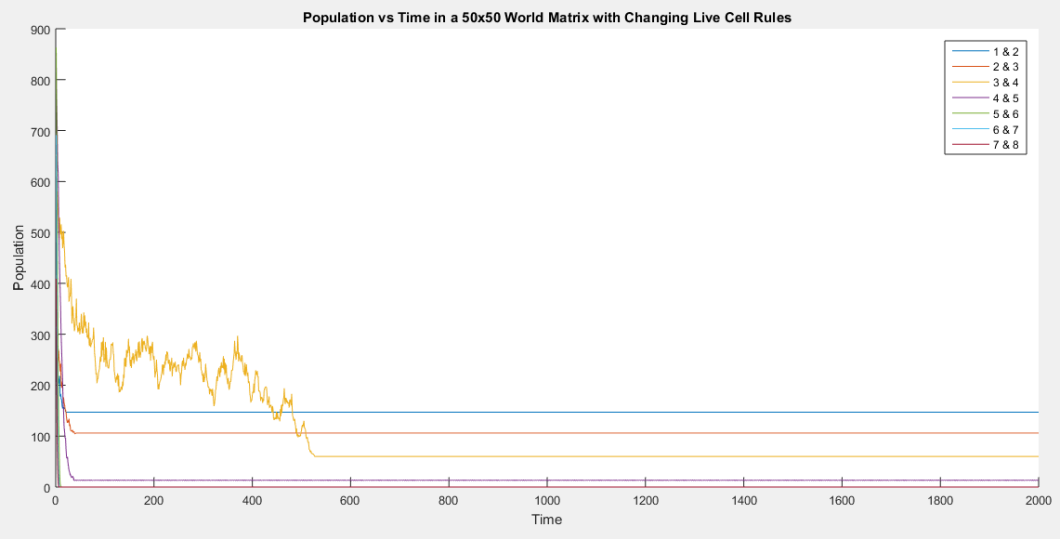

Figure 9: Population vs Time of simulations with various neighbour requirements for live cells as seen in the legend

Figure 9 seems to indicate that a *Game of Life* without the conventional 3 or 4 neighbours needed to live onto the next generation is completely unstable, and plummets from **Stage 1** to **3** equilibrium immediately. The only simulation which had a **stage 1, 2** and **3** was the simulation with conventional rules. The only trend that appeared to occur was that a higher number of neighbours needed to live on to the next generation also gave a higher equilibrium population consistently.

3.4.2 Changing the Rules for Dead Cells

As with changing the live cell rules, changing the dead cell rules also changed the progression of population with time. Here a larger variation of population trends were observed. Requiring that dead cells have 1 or two neighbours in order to become alive caused almost an inverse in population

trends, with population rising and then hitting equilibrium. However, changing the rules of the Game of Life such that 4 to 8 cells were required to turn a dead cell into a live cell simply caused the population to plummet into equilibrium.

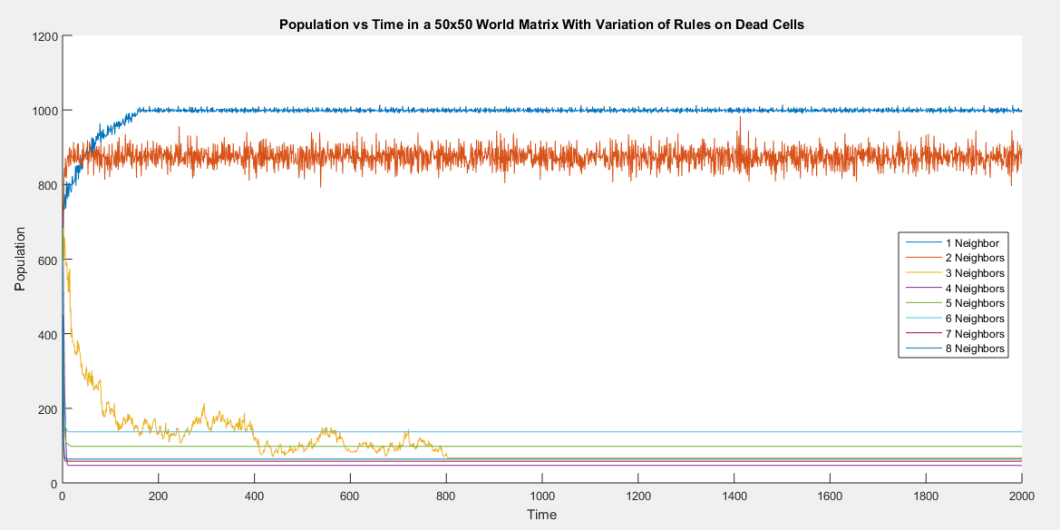

Figure 10: Population vs Time of simulations with various neighbour requirements for dead cells as seen in the legend

3.5 Effect of Changing Population Sparsity on Population Trends

The sparsity, or concentration of live cells in the world matrix can significantly affect the progression of population as a function of time If a world has too low a concentration of live cells, live cells will not have enough world neighbours to replenish the live cell population, and thus will reach **Stage 3** far quicker. Populations with a high concentration will be too overcrowded and reach **Stage 3** quickly as well. Concentrations in the 50% range will achieve a much more typical curve. In order to determine trends on how sparsity affects population, a world matrix was generated, and then using that same matrix, population sparsities ranging from 10% to 90% were generated.

3.5.1 Population vs. Time with a 10-30% sparsity range

Populations with a 10-30% sparsity had a varying range of population trends. A 10% sparsity caused the world population to have a brief **Stage 2**, and go straight into equilibrium, skipping even **Stage 1**. The 20% and 30% sparsity simulations surprisingly resulted in more typically observed population-time graphs, with a **Stage 1, 2** and **3**, although the 20% sparsity simulation due to a lack of cell concentration reached equilibrium hundreds of generations prior to the 30% sparsity simulation. **Figure 11** shows the population vs time of the 10%, 20%, and 30% sparsity graphs.

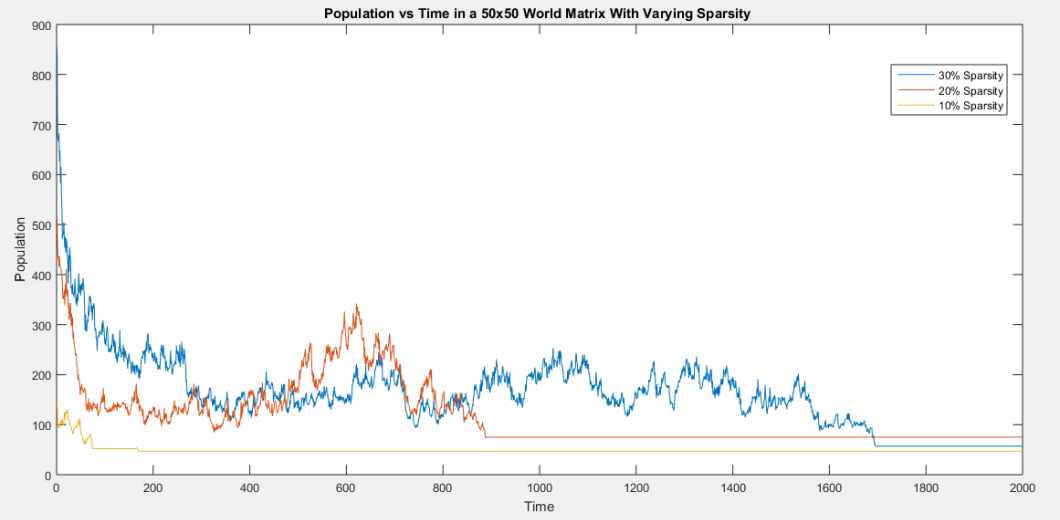

Figure 11: Population vs Time of Matrices with varying sparsities as seen in the legend

3.5.2 Population vs. Time with a 40-60% sparsity range

The simulations with 40-60% sparsities were very consistent, giving very similar population-time graphs to a regular *game of life* simulation. This is to be expected, as the population sparsity of a typical *game of life* is usually in this range. This is because randomly generating the state of each cell in the world will generally lead to around 40-60% of the cells to be alive or dead, which is the sparsity being simulated in this case.

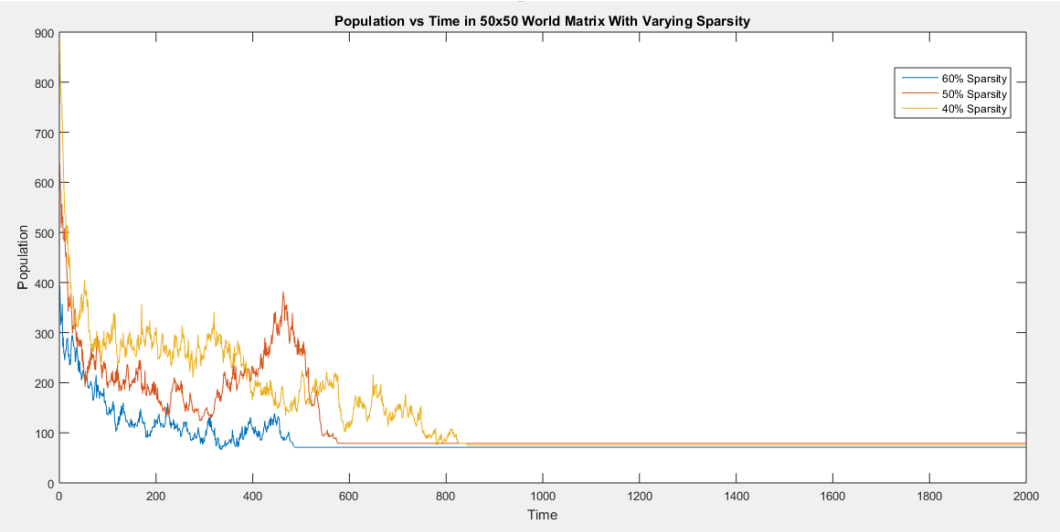

Figure 12: Population vs Time of Matrices with varying sparsities as seen in the legend

3.5.3 Population vs. Time with an 70-90% sparsity range

As can be seen in **Figure 13** the sparsities in the 70-90% range were associated with an immediate transition to an equilibrium state. This makes sense, as a concentration of live cells in that range will not be enough to allow the live cell population to grow appreciably. The only exception to this rule was the 70% sparsity which started at a very spiky **Stage 2**, which was more volatile than a **Stage 2** in a regular simulation, but which also went to equilibrium quite quick.

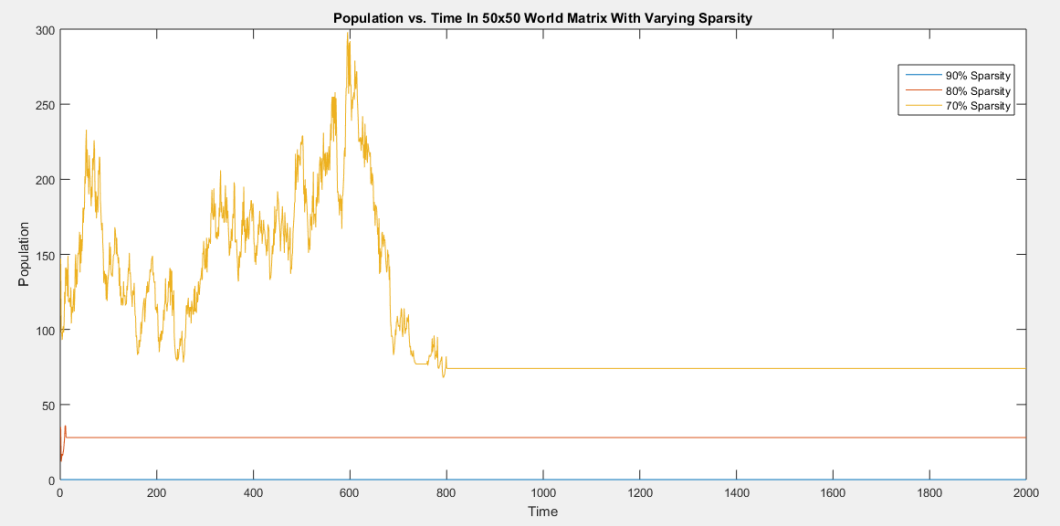

Figure 13: Population vs Time of Matrices with varying sparsities as seen in the legend

3.6 Deriving a Mathematical Expression to Predict Population

While the *Game of Life* is quite volatile and chaotic in nature, making it difficult to make predictions on the state of the population with respect to time, one can formulate a mathematical expression which can predict the progression of the population in a Game of Life. Specifically, one can approximate the equilibrium population as well as the point where **Stage 2** and **3** of the population begins. In order to do this, the data points of a population vs time graph can be taken, and a curve fit can be applied to these points. The data from Figure 2 was taken and curve fitted in Excel, as seen in **Figure 14**.

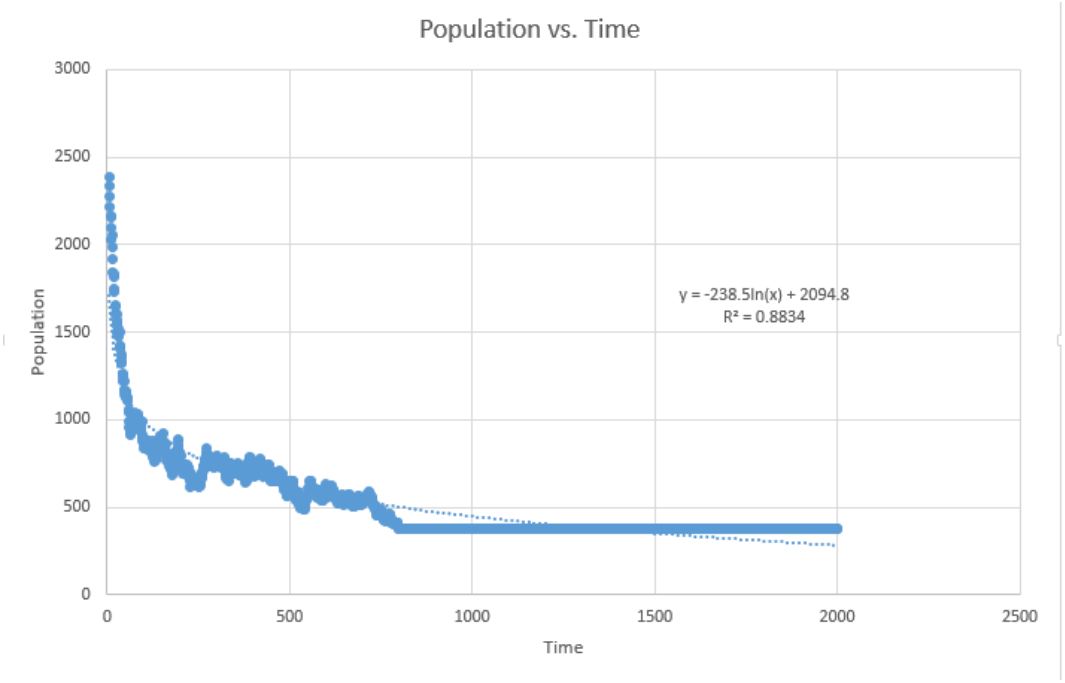

Figure 14: A curve fitted Population vs Time graph with the associated curve fit equation and correlation coefficient R^2

In order to determine what type of mathematical expression best predicts the equilibrium state of a typical *Game of Life*, a variety of equation types, from polynomial to logarithmic were attempted. The equation type with the best R-squared value, which is a coefficient measuring the strength of the correlation between a sample of data and a curve fit, was a logarithmic fit. This was surprising, as the graph of population decay initially appeared exponential. Extrapolating from the equation seen in **Figure 14**, the mathematical expression best able to predict *Game of Life* populations appears to follow **Equation 1**.

$$P(t) = -E * ln(t) + I \tag{1}$$

where $P(t)$ is the population of a world as a function of time, E is the equilibrium population, $ln(t)$ is the natural logarithm as a function of time, and I is the initial population .

3.6.1 Predicting Equilibrium Population

Predictions for the final equilibrium population can be derived by rearranging **Equation 1** into **Equation 2**.

$$\frac{P(t) - I}{ln(t)} = E \tag{2}$$

Once this is achieved, the initial population and first data point can be inputted, giving an approximation for equilibrium population. While this method is certainly exact, it provides an effective approximation. A number of simulations and predictions were run and the mathematical expression was then given data from these simulations and able to make a prediction for the equilibrium population, as seen in **Table 1**.

Table 1: Predicted vs. Actual Equilibrium Population in Varying Matrix Sizes

Matrix Size (nxn)	Equilibrium Population	Predicted Population	Predicted vs. Actual Equilibrium
20	23	41	18
20	7	6	1
40	124	4	120
40	40	45	5
60	105	150	45
60	138	92	46
80	171	160	11
80	312	255	57
100	349	285	64
100	327	371	44

The Predicted vs Actual Equilibrium values varied in accuracy, but in general the predicted value was within 100 cells of the actual value. While obviously not completely accurate, it does further prove that population in the *Game of Life* with conventional rules does progress as stated in **Equation 1**.

3.7 Conclusions

After performing various simulations of the *Game of Life* ad gathering statistical data on those simulations, various conclusions were drawn. Firstly, the progression of the *Game of Life* can be split into 3 stages; An initial stage of volatility and population plummet, a transition to a second stage where the population remains relatively stable and oscillates between periods of increase and decrease in a relatively small downwards trend, and finally, a state of equilibrium where the world is populated only by oscillators. The first and second stage can interchange between each other, reaching a level of stability and oscillation, then plummeting and going down to a period of stability and oscillation once again, until equilibrium is reached.

Adding sources and sinks into a simulation changes the population trend dramatically. If only sinks are present, the population plummets downwards into a state of equilibrium at a rapid rate. Adding an appreciable amount of source cells into a simulation even in the presence of a majority of sinks prevents an initial population plummet and keeps the population relatively stable in a **Stage 2** state, preventing the population from reaching equilibrium. Raising the amount of source cells further prevents the initial population plummet.
Removing periodic boundaries from a graph causes population to progress downwards at a slower rate than in a bounded graph, but in an unbounded graph **Stage 2** is not present, as population trends down without reaching any interval of relative stability. While equilibrium is reached far faster in an unbounded graph, the net equilibrium population is far higher due to the expansion in the world size.
Changing the rules of the simulation also give some interesting results. When changing the amount of neighbours for live cells needed to survive, only the original 3 or 4 neighbours needed to survive resulted in a population which did not immediately plummet into equilibrium. When changing the amount of neighbours a dead cell needed to become alive, two different phenomena are observed. If the amount of neighbours needed for a dead cell to become alive are 1 or 2, population rises and reaches equilibrium, essentially an inverse of the typical population progression which trends downwards into a state of equilibrium. Any neighbour count larger than 3 for a dead cell simply resulted in a plummet to equilibrium.
Varying the sparsity of the graph resulted in a variety of trends depending on the sparsity. If the sparsity was in the 20-60% range, the population progression was typical of the *game of life*. If the sparsity was in the 10%, or 80-90% range, the population experienced a brief **Stage 2** and went straight into equilibrium. A 70% sparsity interestingly gave a spike upwards and then a **Stage 2**, and then a spike downwards into equilibrium, without the initial population plummet, which was an interesting phenomena.
Finally, a curve fit was applied to a sample *Game of Life* simulation, and an equation was derived to approximate population as a function of time, for conventional *Game of Life* rules. This equation was then tasked with predicting the equilibrium populations of a variety of simulations, and it for the most part gave a close approximate value, although it varied in its accuracy.

4 Effect of Introducing Concepts of Evolution and Random Genetic Mutation Into Simulation

The conventional *Game of Life* only has one species moving around a world matrix. However, *Game of Life* situations where more than one species exist in the same world. The concept of aggression and reproduction can also be added, giving rise to a completely different *Game of Life*.

4.0.1 *Game of Life* - Wolves vs. Rabbits

A new game was constructed, called wolves vs Rabbits. In this game, there are two species; wolves and rabbits. Wolves act as the predators, preying on the rabbits, and rabbits are the prey, reproducing and attempting to survive. The rabbits in the game follow the regular rules of the *Game of Life*.

1. If a rabbit has less than 2 neighbouring rabbits it dies from underpopulation.
2. If a rabbit has more than 4 neighbouring rabbits, it dies from overpopulation.
3. If a rabbit has 3 or 4 neighbours it lives onto the next round.
4. If an empty cell has exactly 3 neighbouring rabbits it is born.
5. The rabbits are also time-independent creatures. Every iteration the rabbits are either dead or born at the same time.

Wolves on the other hand are more complex beings.

1. They have an age, a gender, a health, a mother and a father.
2. They have a reproductive rate, an aggression.
3. They are also time-dependent which means that each wolf moves one at a time during an iteration.
4. When the wolf moves it moves randomly by one cell.
5. If the wolf encounters a rabbit it eats it and gains a health benefit.
6. If it encounters a wolf of the same gender that is not it's parent or child, it fights it to the death.
7. During the fight, the Wolf's age, health, aggression rate and the number of wolves is taken into account into its chance of winning the fight.
8. If a wolf encounters a wolf of the opposite sex they will attempt to mate.
9. They will mate as long as they are not directly related (one generation) and their health is not too low.

4.1 Population vs Time of A Wolves vs Rabbits Simulation

Three different simulations were performed for this game type; one where the population of rabbits was far larger than the population of wolves, one where the two populations were equal, and one where the population of wolves was much larger than that of the rabbits.

4.1.1 Simulation with Significantly more Rabbits than Wolves

With a large amount of rabbits and low amount of wolves present in the initial populations, an interesting predator and prey relationship was observed. As rabbit population plummeted, wolf population rose. This makes sense as in the plummet stage the rabbit population is much higher, and wolves have enough food to survive and grow. But as rabbit died out and went into **Stage 2**, the wolves had much less food available, and coupled with a far smaller live rabbit density, could not reach food before dying out. As a result, during **Stage 2** of rabbit population progression, the wolf population died out completely.

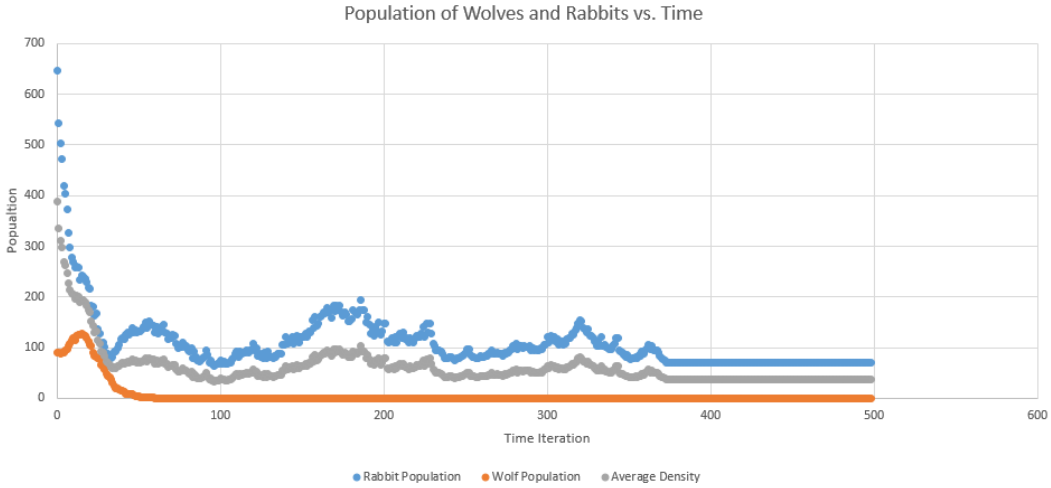

Figure 15: Population and Concentration vs Time of Wolves vs Rabbits simulation with more rabbits than wolves

4.1.2 Simulation with equal Populations of Rabbits and Wolves

In this case the population of rabbits plummeted downwards as they were not only dealing with **Stage 1** of *Game of life* progression,but also with being preyed on by a large amount of wolves. This caused the rabbit population to plummet to zero rather than equilibrium. The wolves on the other hand, remained stable and actually rose in population initially when they had enough prey to remain alive. As the rabbit population plummeted close to zero due to over-predation and the population progression of the game of life, the wolves ran out of prey and spiked downwards at an extremely fast rate into a population of zero, just like the rabbits.

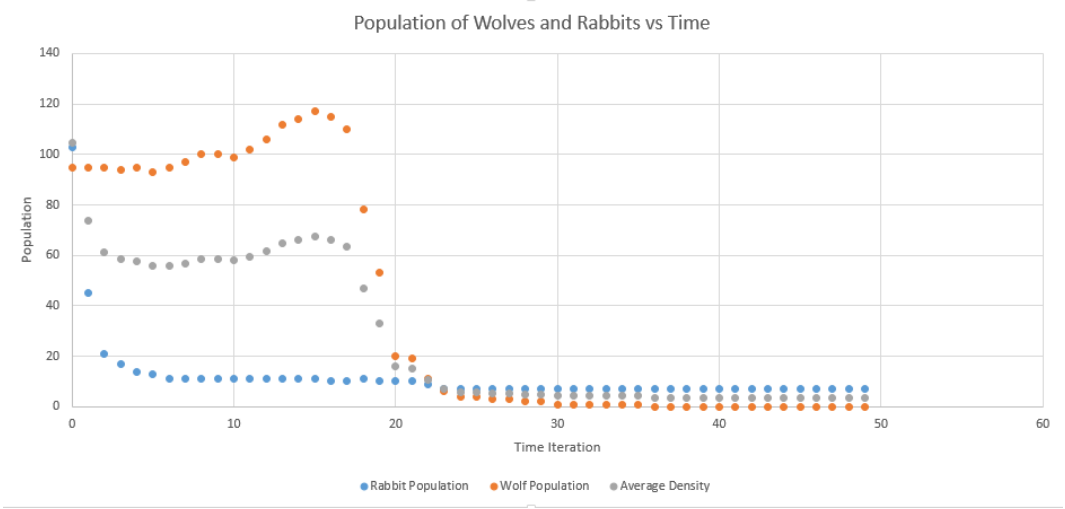

Figure 16: Population and Concentration vs Time of Wolves vs Rabbits simulation with equal amounts of rabbits and wolves

4.1.3 Simulation with Significantly more Wolves than Rabbits

Surprisingly with a much higher population of wolves compared to rabbits, the rabbit population remained stable for a short period of time, until plummeting to zero. With a high initial population and therefore an ability to reproduce, wolves remained stable for a large portion of time until once again spiking downwards into a population of zero abrubtly, due to a lack of food.

4.2 Conclusions on Predator Prey Relationship in Wolves vs Rabbits Simulation

This simulation gave significant insight into predator prey relationships, even resembling those found in nature. When ample prey/rabbit population is present, predators increase in population due to food availability and reproduction. However, due to overhunting the prey population simply dies out, and within a short amount of time thereafter, so do the predators, unable to simply survive by reproduction alone. This rule held for every simulation performed, no matter the predator and prey population ratios.

5 Analysis of Algorithmic Efficiency of the Classical *Game of Life*

A basic *Game of Life* algorithm is quite simple in nature. A program will check the state of each cell, and based on the cell's state it will check the 4 *Game of Life* rules to determine whether the cell will live or die. While this algorithm runs well, it becomes highly inefficient as the size of a

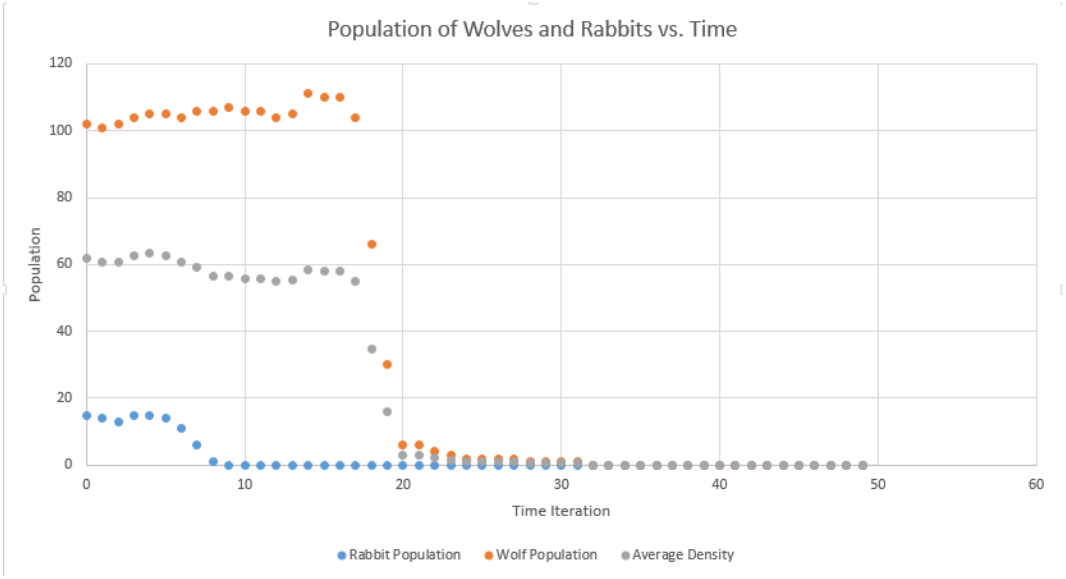

Figure 17: Population and Concentration vs Time of Wolves vs Rabbits simulation with more wolves than rabbits

world matrix increases, due to the fact that it checks every cell in the world matrix. This algorithm can be altered such that a *Game of Life* simulation can not only be run faster, but so that runtime is not as drastically affected by increases in matrix size. A new algorithm, programmed in C, was created and compared to the conventional *Game of Life* algorithm in terms of program runtime.

5.1 A revised *Game of Life* Algorithm

As mentioned before, the conventional *Game of Life* algorithm simply searches through every cell in the matrix and determines the state of each cell for the next generation. This algorithm can be improved significantly if the searching method is shifted from a general search which searches every cell, to a specific search which only checks live cells and its neighbours. This can be done by constructing a list of live cells generated in the initial generation, and then adding or removing elements from that list as each generation progresses, removing dead cells and adding new live cells. The second improvement can be made by also introducing a second layer of the world matrix, which holds how many neighbours each cell has. This second algorithm revision provides the added efficiency of only having to check one element to get the amount of neighbours of a cell, rather than check 8 elements for every cell, bringing the neighbour search for only one cell and its neighbours from 72 to only 8. Furthermore, when a cell dies, all that has to be done is to subtract the value of every cell surrounding it by 1 in th second layer of the world matrix, which is a far shorter and simpler task than performing a search for every cell. For only one cell and its neighbours, the amount of checks goes from 64 to only 8. The algorithm can be seen below.

1. Generate an nxmx2 world matrix
2. Search world matrix for all live points, and store the co-ordinate of every live point in a linked list
3. In the second layer of the world matrix, set the amount of neighbours of every live cell and the amount of neighbours of every cell surrounding a live cell
4. Search through linked list of live cells. For each element in the list, obtain world co-ordinate (i,j).
5. Check (i,j,2), which is where the amount of neighbours of the cell is stored. If (i,j,2) equals 2 or 3 and the cell is currently alive, the cell will live. Otherwise, the cell is to be removed at the end of the world check.
6. Check (i,j,2) value for every neighbouring cell. If (i,j,2) is equal to 3, add cell to new cell linked list, which holds cells which will be added to the live cell list at the end of the world check.
7. Do steps 2-4 for every cell in the live cell list
8. Concatenate live cell list and new cell list. For cells which have been added, set their (i,j,1) value to alive,and add 1 to the (i,j,2) value of every neighbouring cell. For cells to be removed, set the (i,j,1) state to dead. Subtract a value of 1 from the (i,j,2) co-ordinates of every neighbouring cell.
9. One *Game of Life* generation has been completed. Repeat steps 3-7 until the specified amount of generations have been iterated through.

These revisions would provide two improvements. Firstly, the amount of cells checked would be drastically less than in the conventional algorithm, as only live cells and their neighbours are being checked, a much smaller search space than an entire world matrix. Not only this, but search time shifts from being a function of matrix size to being a function of the amount of live cells present in a matrix. This means that as population goes down runtime also goes down since there are less elements to check. This makes this algorithm more and more efficient as generations progress. One drawback however, is the fact that this algorithm is far more memory intensive than the previous algorithm, since two extra lists and an extra world matrix is also generated. Despite this, the aforementioned algorithm improvements should compensate for increased memory consumption.

5.2 Runtime Comparison between Conventinal and Revised Algorithms

In order to test whether the revised algorithm is actually more efficient than the conventional algorithm, both programs were tasked with simulating *Game of Life* simulations of increasing size. The runtime as a function of world size was then graphed. The conventional algorithm was run in C and MATLAB, and the revised algorithm was only run in C. This was done in order to also draw a conclusion about how the language used affects program runtime, as seen in **Figure 18** and **Figure 19**.

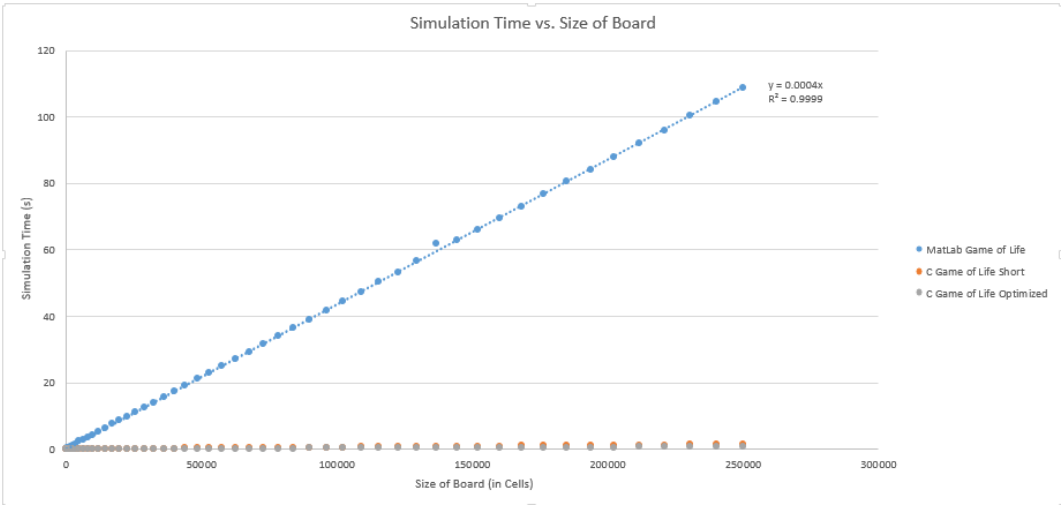

Figure 18: World Size vs Runtime of a conventional MATLAB and C *Game of Life* algorithm as well as a revised *Game of Life* algorithm, in C

After graphing the runtime vs matrix size graphs, a variety of conclusions can be drawn. Firstly, it can be seen that the revised algorithm is in fact more efficient than the conventional algorithm. However, the runtime is only halved, which while being significant is not what was expected. This is likely due to the increased memory consumption as well as the element deletion, addition and searching performed on the linked list, which would also take its own computing time. However, the halving in runtime is still an improvement. What is much more surprising however is how much more efficient the program was when coded in C rather than MATLAB. The conventional algorithm run in C was 75 times more efficient than its MATLAB counterpart, and the revised algorithm was 150 times faster, a drastic improvement in runtime.

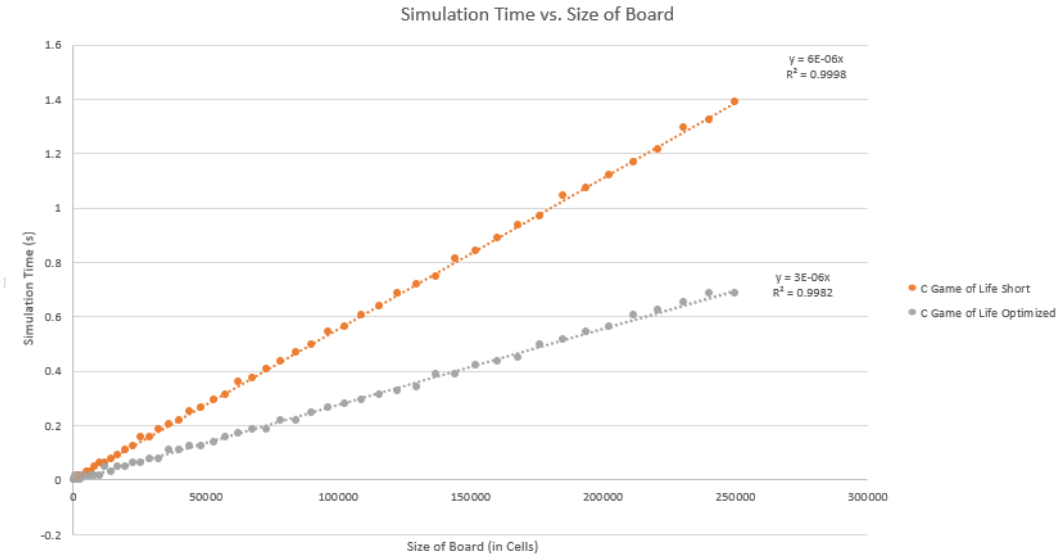

Figure 19: World Size vs Runtime of a conventional C *Game of Life* algorithm as well as a revised *Game of Life* algorithm, in C

5.3 Using Multithreading to Improve Program Runtime

The algorithm revision implemented provided a significant runtime reduction. However, if the computer that is running the simulation has multiple cores, the concept of multithreading can be utilized in order to make runtime even shorter. Multithreading allows a processor with multiple cores to complete tasks simultaneously, with each processor core performing different tasks. This can be applied to the Game of Life for a runtime improvement. While this idea was not implemented due to time constraints, it is still worth mentioning. While this algorithm is unrefined and tasks like thread syncing and so on would have to be performed, it is an interesting idea. In a multithreaded version of the *Game of Life*, there would be searcher and allocator threads. The searcher threads would search through the live cell list simultaneously, passing pointers to reallocator threads for cells which will need to be made alive or dead for the next generation. Cells which need to be killed off would be placed in a list while cells to be made live would be placed in another list. This time once again, each list would have a corresponding thread and each thread a corresponding reallocator. The threads would be synchronized by a checker variable in order to ensure the threads dont access memory and resources simultaneously. The allocator threads would reallocate data to different lists/ structures as requested, and free or allocate memory when needed. Through this method, searching could be done much faster, and be done without having to stop to reallocate memory due to reallocator threads.