ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Lecture with Computer Exercises: Modelling and Simulating Social Systems with MATLAB

Project Report

Circle Of Life

Patrick Misteli & Ruben Kälin

Zurich
May 2014

Agreement for free-download

We hereby agree to make our source code for this project freely available for download from the web pages of the SOMS chair. Furthermore, we assure that all source code is written by ourselves and is not violating any copyright restrictions.

Patrick Misteli

Ruben Kälin

Contents

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Circle of Life

Authored by (in block letters):

For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
Misteli	Patrick
Kälin	Ruben

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
Zürich 12.5.2014	

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.

1 Abstract

In the film "The Lion King" from Disney the main character's father explains to his son the concept of what he calls "the circle of life" and how all animals in the food chain hierarchy are needed in order for every species to survive, [7]. He abstracts the concept by explaining that lions become grass after they die. This grass is eaten by the antelopes, which are again eaten by the lions to complete the circle. We wanted to see if this concept is sound and said three organisms are able to survive within a cellular automaton simulation.

2 Individual contributions

Patrick was responsible for the model, the parameters and the implementation in Matlab thereof. Ruben polished single aspects of the simulation and was responsible for the performance of the algorithm. Furthermore Ruben was responsible for the inclusion of the Lotka Volterra equation and finding papers thereof.

3 Introduction and Motivations

Having a clear food chain hierarchy defined by the given "circle of life", we were interested in whether it is possible to create a stable population cycle that closely relates that of nature by having certain attributes and rules for each organism. Is it possible for the system to remain stable if we were to remove one of the organisms? To the end of the mentioned film "Lion King" the balance is destroyed when the lions start disrespecting the circle of life by overfeeding. Our model should be able to reproduce this effect when making the simulated lions "very hungry". A very popular way of modeling the dynamics in populations is to use the Lotka-Volterra equations, [2]. Since we are using this model as a reference to our model, one of our fundamental research questions is whether it is possible to adjust our model to show similar results as the Lotka-Volterra model.

4 Description of the Model

4.1 States

Starting with Conway's Game of Life [5] a cell should not only have two states (active or inactive) it should now have the following four states representing all organisms:

Nothing or Inactive (Black)
This represents a dead land cell where no living organism is located

Grass (Green)
This represents a living grass organism

Antelope (Brown)
This represents a living antelope organism

Lion (Red)
This represents a living lion organism

4.2 Attributes

Each of the four cell types must have parameterizable attributes in order for the extended rules to work. Table 1 lists all attributes and describes their meaning.

Name [Value Range]	Description
1. Type {1,2,3,4}	The number between 1 and 4 represents "Nothing", "Grass", "Antelope" and "Lion" respectively
2. PreyType {1,2,3,4}	Type of the organism which the current organism eats
3. Becomes {1,2,3,4}	Type to become after a natural death (dying of age or hunger)
4. FoodDigest [0,inf]	Amount of food that is digested in one time-step
5. Stomach [0,inf]	Amount of food currently in organism. This decreases with each time-step (by the number defined in FoodDigest) and increases by 1 when the organism found food
6. MaxStomach [0,inf]	Maximal amount of food an organism will eat. Organism will only eat if its stomach is below this number. A high number will cause the organism to eat anything it finds causing overfeeding while a low number will cause the organism to live on a strict diet
7. DeathProb [0,1]	The probability of dying by age in the beginning of a time-step. Other death causes are discussed in Section 4.5
8. Fatness [0,inf]	The number of predators that can be fed eating this organism (Note: as soon as one predator takes a bite the current organism will become Nothing at the end of the time-step)
9. Alive {0,1}	1 if alive, 0 if a predator has bitten this organism
10. MinStomachRep [0,inf]	Minimal value of Stomach needed to reproduce (Reproduction explained in Section 4.3.6)
11. ReproductionProb [0,1]	Probability to reproduce when allowed (Reproduction explained in Section 4.3.6)
12. IsOffspring {0,1}	1 if newborn, 0 if adult

Table 1: A description of all attributes that define an organism.

4.3 Protocol per Timestep

Once we have generated an initial map with random cells we start the time process. This means for an unlimited amount of time steps every cell gets called in random order and completes the time-step protocol. This protocol is very similar to the rules in Conway's Game of Life but has various extensions in order to simulate the desired behavior of nature. The following is a detailed description of what each cell processes in one time-step. Figure 1 visualizes this process further.

4.3.1 (1) Initial check

In the beginning of the protocol we check whether the *Alive* attribute is still set to 1. If it is set to 0 it means at least one predator cell has taken a bite of the current organism in this time-step. This corresponds to a predator killing its prey in nature. If the the current cell is declared as dead it cannot (naturally) do anything anymore. We thus skip the protocol from here on and initiate the protocol on a next cell. We do not remove the organism yet, since more predators may be within the neighborhood and feed on this organism.
We furthermore check whether the current cell is an offspring ($IsOffspring == 1$). This would mean 2 other cells of the same type have generated this cell in this time-step. A newly born cell is also unable to do anything and we therefore would skip the rest of the protocol from here on. In nature, a newly born animal is able to neither reproduce nor hunt directly. We are aware that there are animals who are able to hunt directly "out of the shell" (such as crocodiles), but we are dealing with antelopes and lions in our simulation which do not posses such skills. In addition, this attribute prevents a chain reaction of organisms generating many generations of children in one step. For example: Two lions produce an offspring in an empty field next to them. The offspring would then be able to reproduce with one of its parents in a cell next to it. Hyperpolating this will result in a chain of newborn lions in only one time-step.

4.3.2 (2) Livability Check

We now subtract number defined in the digestion attribute from the content of the organisms stomach. If this causes a negative number in the stomach of the organism has died of hunger and is instantly replaced by the type defined in its "Becomes" attribute. Furthermore, we generate a random number. If this number is smaller than the death probability attribute the organism has died of age. As having died of hunger the organism is instantly replaced by the type defined in its "Becomes" attribute.
If the organism has died we skip the rest of the protocol and move on to the next

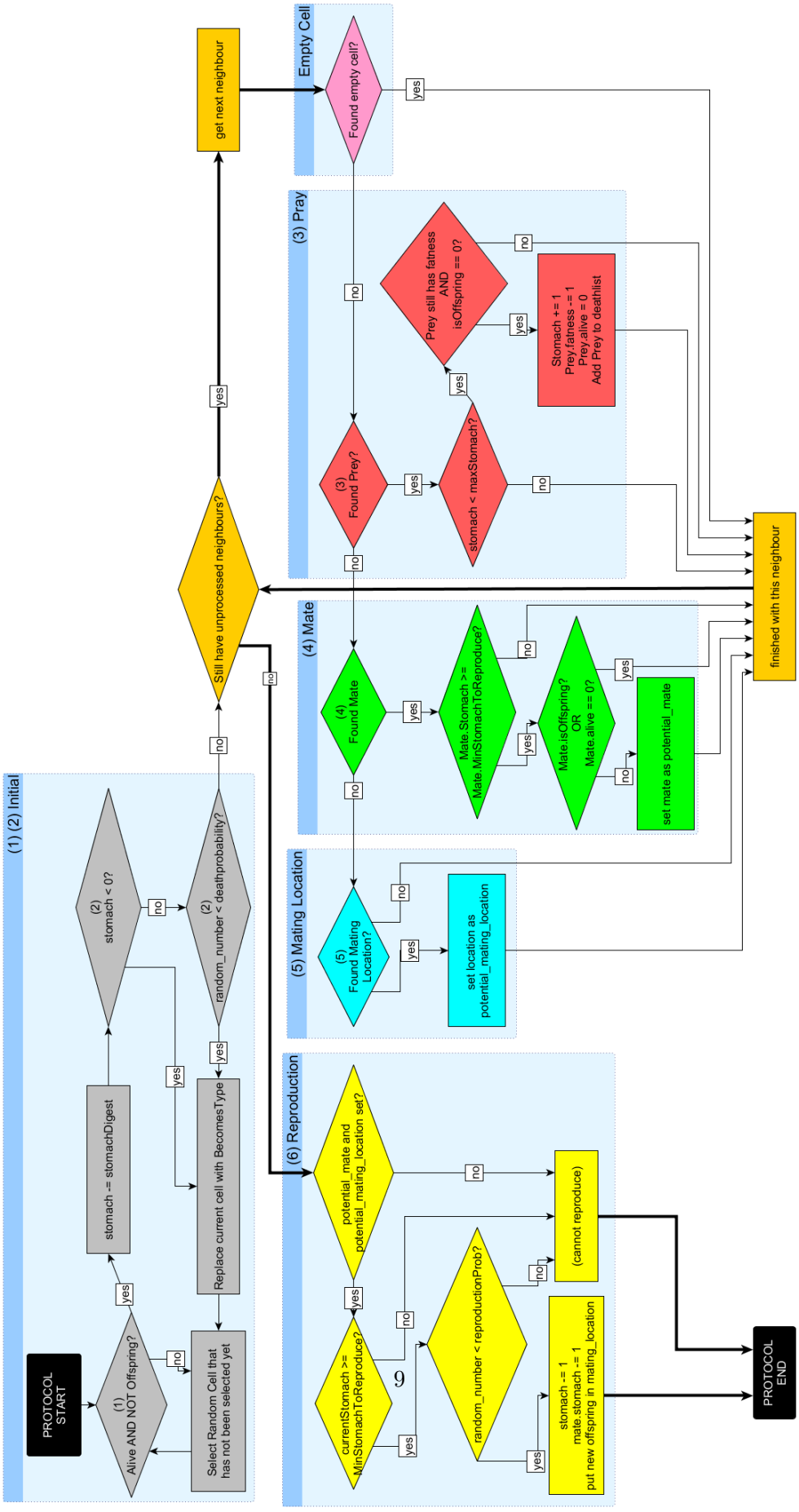


Figure 1: Protocol for each time-step

cell. If the organism is still alive we can move on to the next protocol step. From here on we consider the neighborhood around the cell and act accordingly. The neighborhood is further described in Section 4.4.

4.3.3 (3) Found Prey

If we find an organism within the neighborhood who's type matches the type which was set in the current cell's PreyType attribute we have found food. We check if the current organism is not already full (has eaten enough) by testing whether its stomach is lower than its MaxStomach attribute. We then check whether the pray still has *fatness* left by checking its fatness attribute. A prey can only feed a fixed amount of predators. This amount is stored in the fatness attribute. The attribute is used to limit the number of predators who can eat this organism. Setting it to 1 means only one predator is able to feed himself before the prey becomes inedible. This corresponds to antelopes or grass only being able to fill the stomach of a certain amount of lions or antelopes respectively.
If the organism we encountered still has fatness larger than 0 we further check whether the prey is an offspring. Being an offspring in a time-step means the organism is busy being born. We thus assume full protection of its parents (who are in turn edible themselves) making them inaccessible for predators.
If all the pre checks return positive we are able to eat the prey. We do this by adding 1 to the current organism's stomach and make sure the *Alive* attribute of the prey is set to 0 to incapacitate it. If this was not the case before we then add the organism to a "deathlist". This list is used to keep track of all eaten organisms and after each time-step all organisms on the deathlist are removed (i.e become Nothing).

4.3.4 (4) Found Mate

If we find an organism who's type is equal to the current organisms type we check if it is possible to mate with it. This is done by ensuring the neighbor organism is alive (*Alive* attribute == 1) and that it has a high enough *Stomach* attribute in order to reproduce. I.e. the *Stomach* attribute must be equal or higher to the amount of the *MinStomachRep* attribute. Furthermore we ensure that the neighboring organism is not an offspring since a freshly born organism is unable to reproduce. We are aware that there exist animals that are able to reproduce within the first day they are born (for example a *Caenorhabditis Elegans* [4]), but antelopes and lions are not amongst those. If all these conditions are met the neighboring animal is set as a potential mate. An organism can only have one potential mate per timestep in order to keep a realistic reproduction rate.

4.3.5 (5) Found Mating Location

If we find either a Grass or a Nothing cell in the surrounding neighborhood we mark it as a potential mating location. A Nothing cell receives higher priority in order not to replace potential food for the antelopes.

4.3.6 (6) Reproduce

Once we have processed all the cells in the neighborhood we now survey the outcome of the mating situation. If we have a potential mate and a potential mating location we can continue with the final mating protocol. This is done by checking if the current organism's *Stomach* attribute is equal or higher than the amount of its *MinStomachRep* attribute. If this is also the case it means there is a potential mate who is enough fed, a cell within the neighborhood that can be replaced by an offspring, and the current organism is fed enough to reproduce. For a final test we generate a random number between 0 and 1. Reproduction is only possible if this random number is smaller than the *ReproductionProbability* attribute of the organism type. This simulates natures success rate of giving birth. Having a reproduction probability of 1 would result in 100 % success rate given the previous conditions are met. When the two organisms successfully reproduce the stomach of the offspring is set to the average of its parents plus 1. This reflects natures way of disabling two underfed parents to reproduce a well-fed child and therefore creating already eaten food. In addition the two parents both lose 1 stomach point in order to keep reproduction to a limit. This should simulate the increase of life difficulty of an organism once it has produced an offspring. Reproduction is natural, but it also hinders the parent's flexibility in life. Subtracting one stomach point enforces this concept, since the organism must now work harder (eat more) in order to stay alive.

4.4 Neighbourhood

As in game of life we inspect our neighborhood, see what kind of cells we find and act accordingly. In the beginning, we used the standard Moore-neighborhood and started testing an extended neighborhood later. This extension allows an organism to "see" further in the map and therefore, not only eat more but reproduce with organism further away. Effects of this are discussed in Section 7.2.6

4.5 Death Causes

In our model there are multiple ways a single organism can cease to exist. Depending on the cause the resulting cell differs too. Table 2 shows all possible death causes.

Death Cause	Description
Age	At each time-step a random number between 0 and 1 is generated. If this number is smaller than the specified death probability the organism dies of age and the cell is turned into the type that is set in its *Becomes* attribute
Hunger	When an organism's stomach reaches a negative number it dies of hunger and the cell is turned into the type that is set in its *Becomes* attribute
Eaten	An organism can be eaten by its predator (or multiple predators in one time-step). If this is the case the cell becomes Nothing after the time-step is completed. We chose not to let the cell become the type that is set in its *Becomes* attribute because after being eaten an organism mainly resides in the stomach of its predator and does not contribute to the outside land anymore

Table 2: All death causes considered in our model.

4.6 Known limits

Our model has certain limits due to abstraction of the concept. One major limit is the inability of our organisms to move. In nature a herd of antelopes would escape the danger if detected. Since our organisms are unable switch places with another cell, the only way to move is by producing an offspring in a cell within the neighborhood. This of course requires a second organism of the same type. Thus, a single organism is unable to move while multiple organisms may create the illusion of moving. This does not differ much from Conway's Game of Life, where cells are also unable to move but patterns (such as a "glider", [5]) create the visual illusion that it is moving across the land.

5 Implementation

We used Matlab to create our simulation and an X-by-Y-by-12 sized matrix to store the data where X and Y correspond to the size of the map (Discussion about map sizes in Section 7.2.5). Having a 3d matrix allowed us to keep multiple attributes for each cell. Each layer of the 3d Matrix corresponds to one of the 12 attributes. For visual representation we used the following four subplots:

- A color coded 2d matrix of the current land. This was obtained by taking the matrix at level 1 of the 3d matrix, which shows the type of organism for each location on the land

- A stacked bar plot to show the amount of deaths for each organism and the cause thereof.
- A second stacked bar plot showing the previous stacked bar plot with normalized numbers. This gives a better view of the percentages of each death cause for each organism independent of the total number of deaths.
- A line plot with a line for each of the 3 organisms (Grass, Antelope, Lion). The x-axis represents the time where the unit is "timestep" and the y-axis represents the quantity of the organism. This graph shows the current population on the land.

The final plot is shown in Figure 2.
In order to disable an organism to be cornered by its predator we implemented the map with a wraparound, allowing an organism on the edge to reproduce or eat to/from a cell on the other side of the land.

6 Lotka Volterra

The population of different organisms has been thoroughly studied. A.J. Lotka and V. Volterra found formulas describing the population number of two species, [6], [8]. One species is a predator and the other species is the prey. A large number of predators results in a decrease of prey organisms and a small number of predators allows prey organisms to increase in number. On the other hand, the predators are fed by the prey, so predators can only flourish when there are enough prey organisms to eat. When the number of prey organism becomes too little, predators die of hunger. These rules are reflected in the formulas of Lotka and Volterra, which show the dynamics of the population, [2].

$$\begin{aligned} \frac{dx}{dt} &= ax - bxy \\ \frac{dy}{dt} &= cxy - dy \end{aligned} \tag{1}$$

The reasoning about the population count of organisms has been extended to three species forming a food chain, [3]. In the model, organisms Z are predators of organisms Y and organisms Y prey on organisms X. Therefore, compared to the two species situation, the population count of organism Y now additionally depends on

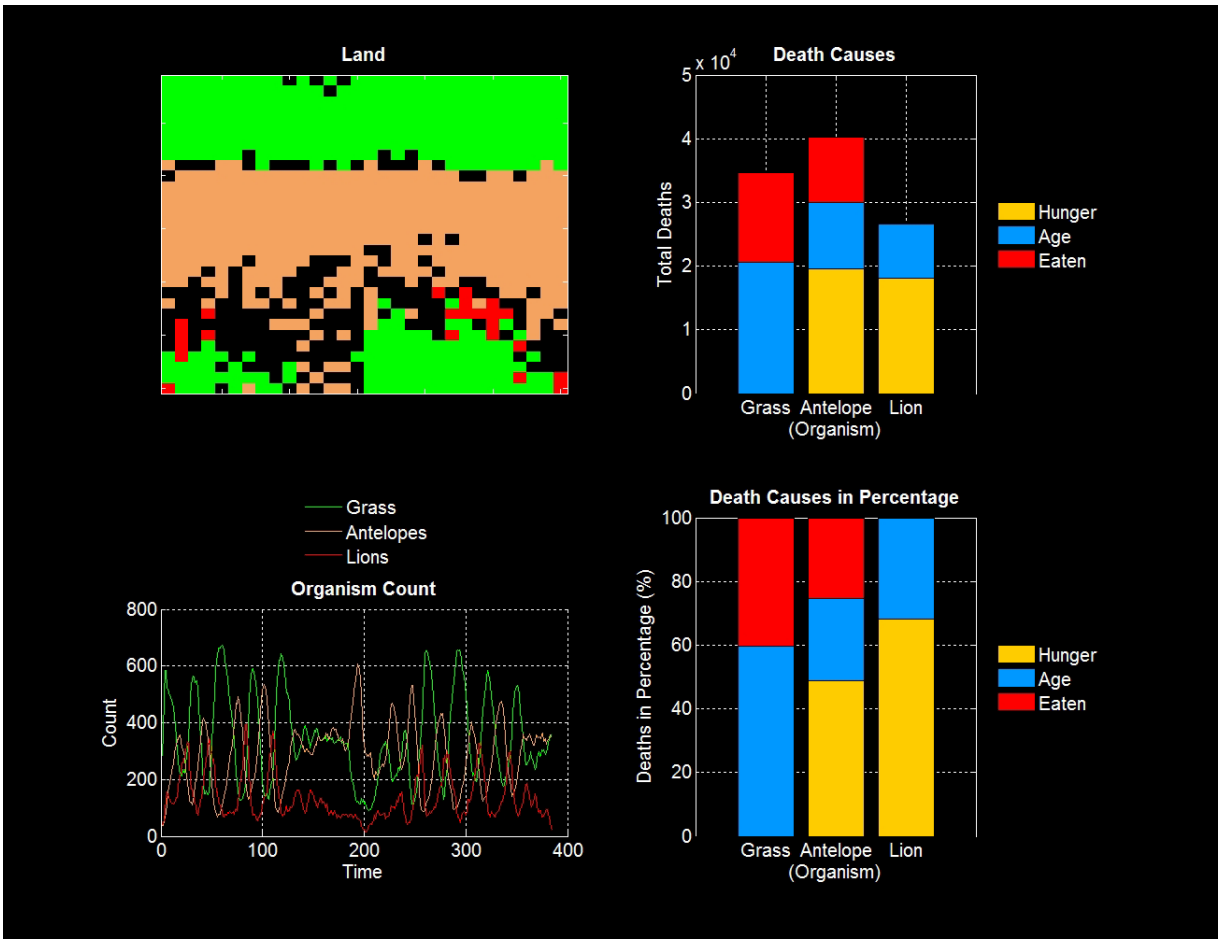


Figure 2: Four subplots

the number predators (organisms Z).

$$\begin{aligned} \frac{dx}{dt} &= ax - bxy \\ \frac{dy}{dt} &= -cy + dxy - eyz \\ \frac{dz}{dt} &= -fz + gyz \end{aligned} \tag{2}$$

The resulting curves can be parametrized by the initial population counts $START_X$, $START_Y$, and $START_Z$ as well as the six constants a,b,c,d,e,f, and g as shown in Table 3. This model could be further generalized in a similar manner to model n

Parameter	Description
$START_X$	the initial number of organisms X
$START_Y$	the initial number of organisms Y
$START_Z$	the initial number of organisms Z
a	reproduction rate of organisms X
b	disadvantage of organisms X of being hunted
c	vulnerability of organisms Y to overpopulation
d	advantage of organisms Y of hunting
e	disadvantage of organisms Y of being hunted
f	vulnerability of organisms Z to overpopulation
g	advantage of organisms Z of hunting

Table 3: The semantic meaning of the different constants.

species. The behaviour of the population curves highly depends on the choice of variables and is thoroughly studied in [3]. The variable assignment $a = b = c = d = e = f = g = 1$ with the initial populations $START_X = 0.5, START_Y = 1, START_Z = 2$ results in the oscillating population count curves shown in Figure 3.

7 Simulation Results and Discussion

Running multiple simulations and adjusting the parameters and the initial map, we discovered several dependencies. Additionally we compared the population results obtained from the simulation with the Lotka-Volterra population curves detected strong similarities.

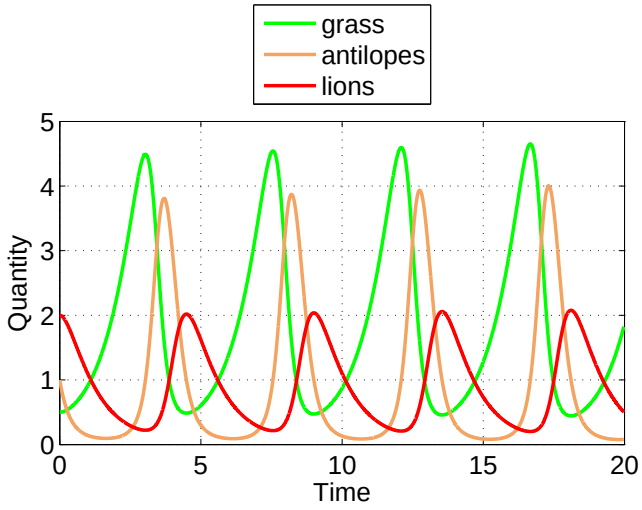


Figure 3: Lotka-Volterra equations for $a = b = c = d = e = f = g = 1, start_X = 0,5, start_Y = 1$, and $start_Z = 2$.

7.1 Chosen Parameters

A possibility of a simple parameter combination that produces a stable or oscillating population is given in Table 4. The death probability was calculated using the helper function $deathProb(a) = 1 - (\frac{a}{a+1})$ which takes input a = average life expectancy and outputs the corresponding death probability.
We have decided NOT to let antelopes become grass when they die of age or hunger but rather become nothing. The reason for this is the dedication to a *Circle* of life. Antelopes get eaten by lions which become grass after they die which can then again be eaten by the antelopes. If antelopes were to become grass too after passing away there would be an additional arrow from antelope to grass. For the purpose of our model we decided against this addition. For simplicity we have left the minimum stomach to reproduce at 0, which means a living organism will always be fed enough to reproduce an offspring.
Using our attribute parameterization from Table 4 in a randomized 40 by 40 land generates the population curve depicted in Figure 7.
Adjusting the values in the Lotka-Volterra formulas for three species to the values shown in Table 6 a plot shown in Figure 8 can be generated. There are some clear similarities visible when comparing the two resulting graphs. The graph from our simulation is less smooth and will not oscillate forever due to the randomness in our model and the location dependency (discussed in Section 7.2.3). Clearly visible are

the three peaks of population from each type in reverse order of the food hierarchy with peaks getting smaller the higher we move up the food chain.

Attribute Type	Nothing	Grass	Antelope	Lion
1. Type {0,1,2,3}	0	1	2	3
2. PreyType {-1,0,1,2,3}	-1	-1	1	2
3. Becomes {0,1,2,3}	0	0	0	1
4. FoodDigest [0,inf]	0	0	$\frac{1}{2}$	1
5. Stomach [0,inf]	∞	∞	10	10
6. MaxStomach [0,inf]	0	0	10	10
7. DeathProb [0,1]	∞	$\frac{1}{6}$	$\frac{1}{11}$	$\frac{1}{11}$
8. Fatness [0,inf]	0	8	8	0
9. Alive {0,1}	0	1	1	1
10. MinStomachRep [0,inf]	∞	0	0	0
11. ReproductionProb [0,1]	0	0.7	1	1
12. IsOffspring {0,1}	1	1	1	1

Table 4: The chosen attributes for each organism.

Attribute Type	Nothing	Grass	Antelope	Lion
4. FoodDigest [0,inf]	0	0	$\frac{1}{5}$	$\frac{1}{5}$
5. Stomach [0,inf]	∞	∞	2	2
6. MaxStomach [0,inf]	0	0	2	2
8. Fatness [0,inf]	0	2	8	0
10. MinStomachRep [0,inf]	∞	0	1	1
11. ReproductionProb [0,1]	0	0.7	1	0.6

Table 5: The changed attributes from table 4 to adapt organisms to a larger neighborhood.

7.2 General Observations

Our simulation is based on a set of inherently local rules. Two organisms only influence each other when both are in each others neighborbood. Therefore, the fastest possible propagation speed is one times the radius of the neighborhood per time-step. However, structures larger than one neighborhood can emerge over time when the influence spreads across the land. These macro phenomena seem to obey their own set of rules even if they are only the interaction between many cells. In this section we introduce the patterns encountered during the simulations and explain why they naturally emerge.

7.2.1 Oscillating or Stable population

There are two types of population variation. A stable population is one where all organisms remain their numbers. It is important to note that organisms still reproduce and pass away, but the rate of reproduction and dying are almost equal at every time-step.
The second type of population we discovered was an oscillating population. Here the ratio from death rate to birthrate oscillates from one side to the other. Populations tend to oscillate because the different species depend on each other. Lions need to eat antelopes, antelopes need to eat grass and grass needs to have space to grow or be created by the deaths of lions. At some point the grass population starts to grow as there are not enough antelopes to eat them. The more grass there is the better for the antelopes because they can feed themselves and their offspring. This results in a decrease of grass organisms and an increase in antelopes. An increase in antelopes means a better life set up for lions since now they are able to feed themselves and their offspring. This results in an increase of lions and decrease of antelopes. Having no food left, the lions start dying and becoming grass. This results an increase of grass since not only the lions generate it but there are little antelopes left to prevent the growth of the grass population. More grass and less lions is a good life set up for antelopes to start growing again and so forth.
This oscillation can have different sizes. A large size of oscillation means animals go from overpopulation to almost extinction. A very small oscillation could be regarded as almost stable. If the oscillation is too large one organism takes over and wipes the entire land. Further reasons for "wipeouts" of the land are the neighborhood size (discussed in Section 7.2.6) and the size of the land (discussed in Section 7.2.5).
In addition it is possible that an oscillating population will find an almost stable state, while an almost stable state may start oscillating again. Figure 2 shows an oscillating state between the time-steps 0-120. An almost stable state follows till time-step 180 after which the oscillation is re-initiated.

7.2.2 Patterns

As in Conway's Game of Life there are certain patterns that appear when we start with a randomized land. This is the case even though the selection and update of each cell and the selection of its corresponding neighbor happens in random order. At first there is a strong oscillation until small groups (or "herds") of the same organism have formed. At the beginning the location is not a big factor since all animals are placed randomly on the map and have random neighbors. Thus the behavior is much like the one described by the Lotka-Volterra equation. After a while organisms of the same type that were close to each other at the beginning start forming a herd,

while lonely organisms die out because of the incapability to reproduce on their own. The following two patterns were observed

- **Chased Antelope Herd:** An antelope herd is formed as described above. A lion herd can "dock" against one side of this herd. This results in antelopes only being able to reproduce on one side of the herd where on the other side of the herd the lions eat the antelopes and produce new offsprings in the now empty cells. If an antelope would produce an offspring in a cell of an eaten antelope it would most probably get eaten within the next time-step by a neighboring lion. The side of the lion herd that is not facing the antelope herd die out because of lack of food. This produces two adjacent herds where one side is well nourished and can reproduce (i.e. move) while the other side has no food and/or predators eating them. Figure 4 shows this pattern.
- **Lion Island:** A lion island is formed when a lion herd is surrounded by antelopes. This means the lions can reproduce in all directions and thus expand the herd quickly. Since the grass generated by lions passing away is surrounded by well-fed lions it is inaccessible for the antelopes (unlike the previous "chased antelope herd" pattern). In the extreme case this could lead to a wipeout of all animals. Figure 5 demonstrates this effect. We start with an initial land shown in Figure 5a. The antelopes start overfeeding while only two small lion herds remain. In Figure 5b the lions are completely surrounded and start expanding. 5c shows that the only grass on the map is guarded by a circle of lions and thus inaccessible for antelopes. Figure 5d shows the consequence of this. The lions have eventually eaten all antelopes and are dying out because of hunger and age. The map size is important for this effect to wipe out the land. This is discussed in Section 7.2.5.

7.2.3 Location Dependency

The main difference between our model and the Lotka-Volterra equation is that Lotka-Volterra only takes into account the quantity of the population, but not the location thereof. In a land where antelopes and lions are separated too far away it is not possible for them to have any effect on each other. This was already mentioned as a known limit of our model (Section 4.6). In our simulation we experienced extinctions of a species simply because they were unable to reach its food source in time before dying of age or hunger. Species may not only be unable to reach a food source but may also be unable to reach a potential mate, which makes reproduction impossible. When having one species gone extinct it is usually only a matter of time until another species will go extinct as well. This is further elucidated in Section 7.2.7.

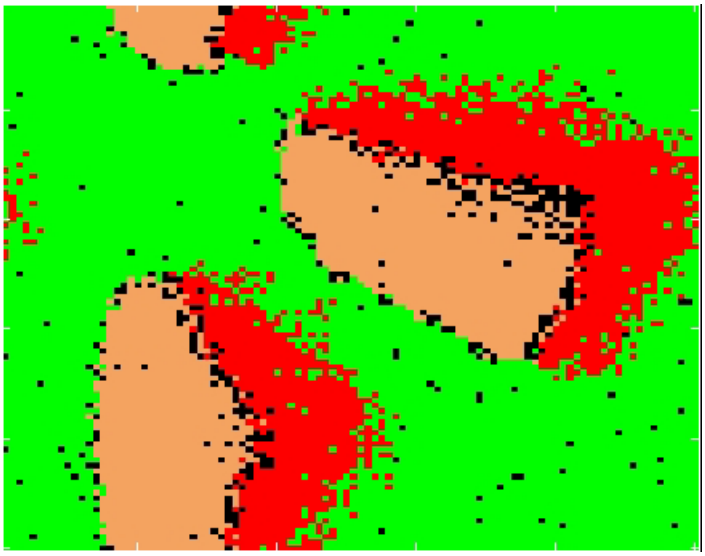


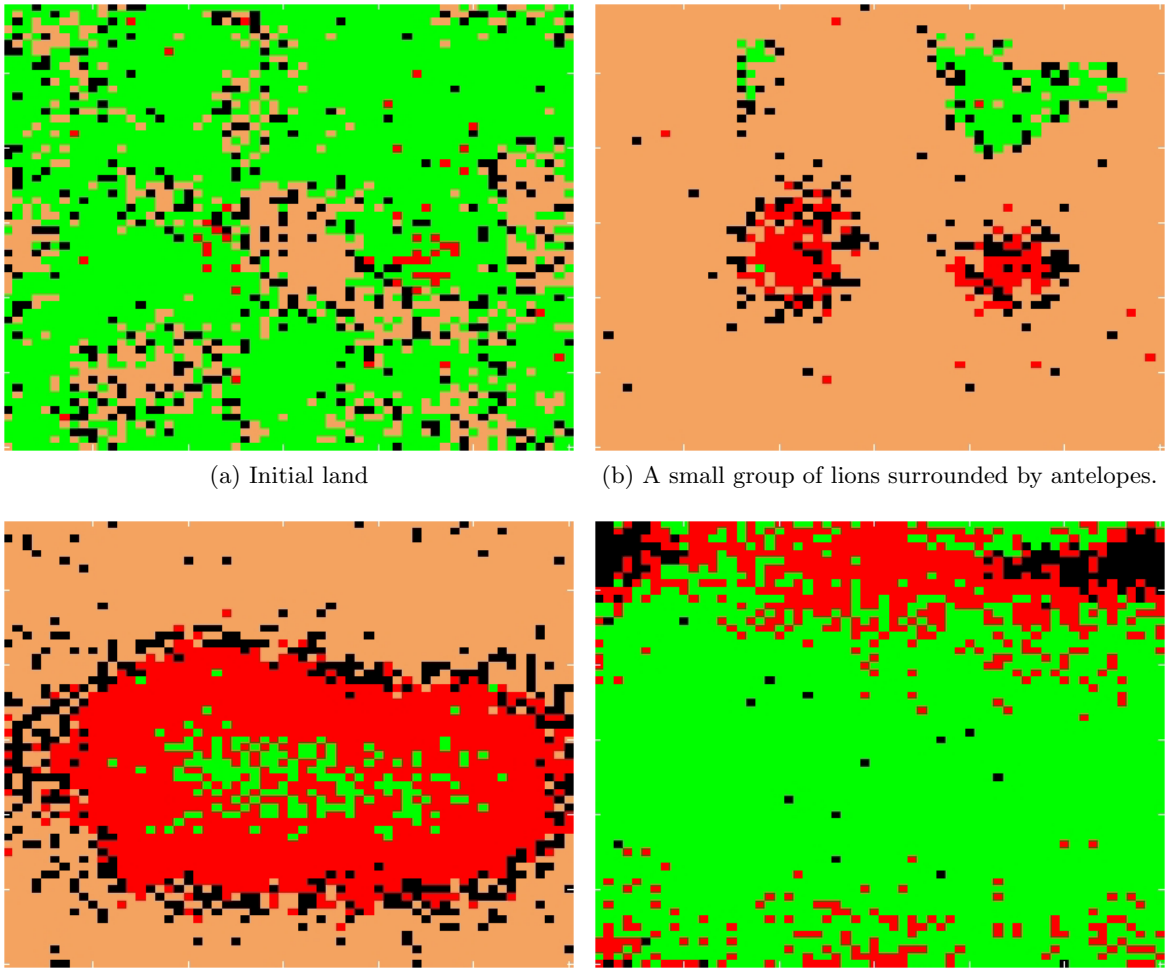
Figure 4: Lions chasing a herd of antelopes.

7.2.4 Random Selection vs Sequential Selection

We have designed to randomly select a cell in one time-step to update while making sure that every cell gets updated exactly once per time-step. Furthermore the selection of the neighboring cell is also randomly selected from the defined neighborhood. In an initial attempt we had both selections running in sequential order. This caused certain patterns to always move to the top left since those cell are updated first. By being updated first they are able to feed first before the others get a chance. Furthermore if no empty cells are in the neighborhood, the first grass cell on the top left will always be selected to turn into an offspring. By making it random we ensure that no predefined pattern and pattern-direction can occur due to the updating order.

7.2.5 Map Size

As mentioned in Section 7.2.3 the location and surrounding of each organism is vital for its survival. Thus the size of the land plays an important role. A smaller map enables a single organism to have everything in its neighborhood and act accordingly. This especially makes life temporarily easy for lions since they will always detect potential food and mates in its neighborhood. Lions taking over in a small map is therefore very likely if not tamed by its parameter.
Oscillating in a small land is difficult to achieve because not much freedom is allowed. As soon as one organism takes over it could mean the extinction for another one due



(a) Initial land

(b) A small group of lions surrounded by antelopes.

(c) Surrounded lions start expanding and prevent antelopes from reaching grass.

(d) Extinction of antelopes can result from surrounded lions.

Figure 5: Various emerging patterns.

to little space left and due to inability to escape its predator.
Having a larger land a Lion Island pattern (Section 7.2.2) does not guarantee the extinction of antelopes. The reason for this is that a herd of antelopes could exist with a distance great enough for the lions not to reach them. As soon as the lions run out of antelopes to eat they start dying of hunger and the protecting lion-circle around the grass is broken allowing the distanced antelope herd to enter.
Furthermore, oscillating structures have a fixed size dependent on the chosen neighborhood size, since each organism only considers the cells in its neighborhood. A larger map can fit more of these structures on it. If an oscillation becomes unstable it could lead to a local extinction and/or overpopulation. Local extinctions can be canceled out by another local overpopulation of the same organism on the same land. The larger the map the higher the probability of this happening. Figure 6 shows a 1000 by 1000 land. It is clearly visible that a stable non-oscillating population is present. Smaller lands may only fit one oscillating structure on it or even only parts of it. This increases the chance of an organism to go extinct on the entire land since there is no room for a second oscillation to cancel this out.

7.2.6 Neighborhood Size

In our model we have chosen the Moore neighborhood and extended it by adding one more field to the northern, eastern, southern and western neighbor, [9]. The resulting neighborhood is also known as the "Von Neumann neighborhood" with a Manhattan distance of 2, [10]. In other terms we have produced a 12-connected-neighborhood which seemed an optimal viewing depth for each organism with our chosen parameters. Decreasing this relates to a slightly blind organism. It cannot see the possible food or mate around it and will therefore die quicker of hunger without being able to reproduce before. Increasing this relates to giving each organism super-vision of the land. A predator is able to eat another cell further away. Which means the escaping of one species from its predator gets harder. The simulation with our chosen parameters showed that an organism who is able to see the next three neighbors (48-connected-neighborhood) already affects the oscillation of the population in a way such that extinction of one species is almost guaranteed. This happens because our parameters allow an organism to eat until its stomach is full (MaxStomach = 10) and reproduce until its stomach is empty. Therefore, each organism is able to reproduce with up to 10 organisms that choose the observed organism as a mate in a single time-step. This may cause overpopulation and result in extinction (further explained in Section 7.2.7). Note, that an organism can still only actively initiate one reproduction per time-step. Adjusting the parameters to allow a stable population with a 48-connected-neighborhood is possible. The adjusted values are shown

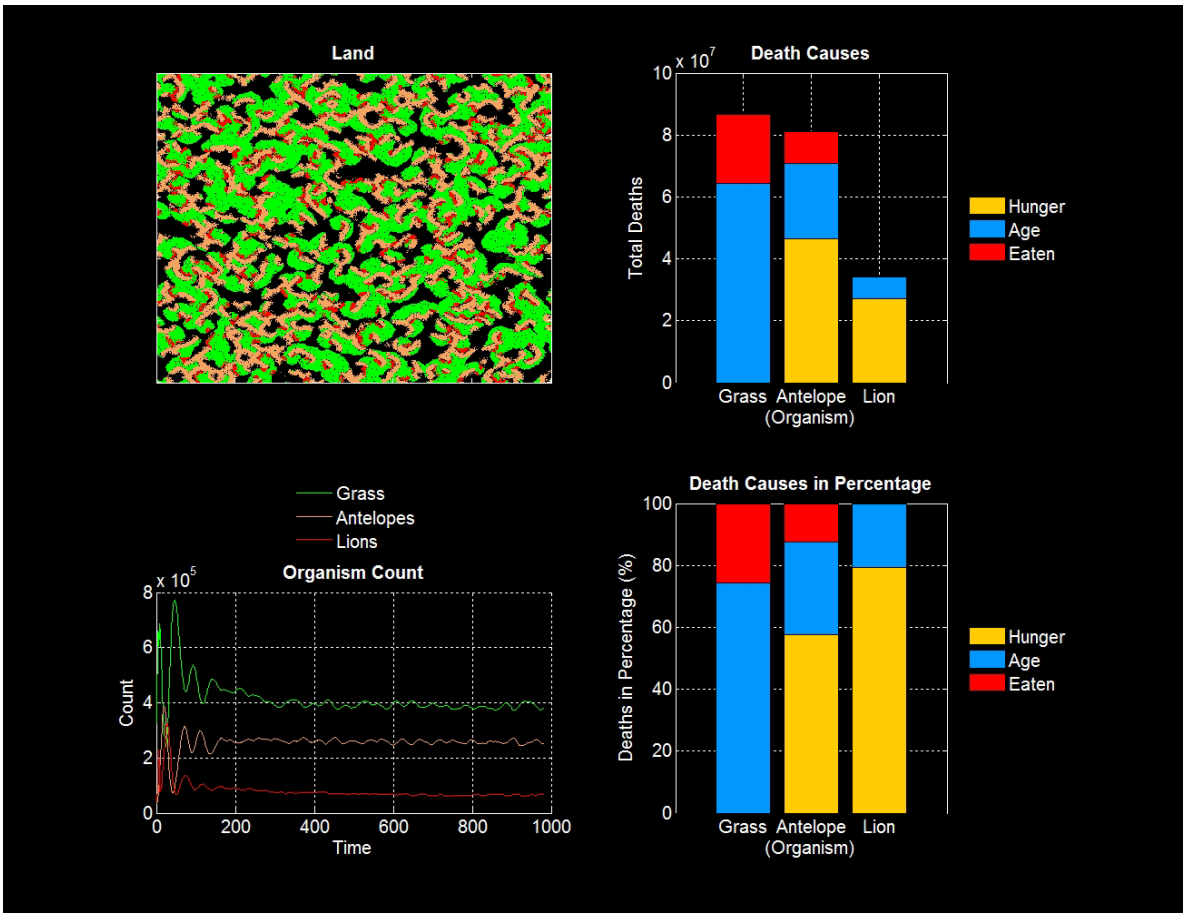


Figure 6: Circle of life simulation with large land 1000x1000 cells.

in Table 5. We adapted the food digestion speed in order for animals to live longer without the need to feed. We also lowered the initial Stomach and MaxStomach to prevent lions and antelopes to overfeed. The fatness of grass was also lowered to only allow 2 antelopes to feed from one grass cell. Lastly the reproduction was adjusted by setting a higher minimum stomach and a lower reproduction probability in order to make producing offspring less likely and preventing an overpopulating of the land. An interesting aspect of the new setup is that it does not work for a smaller neighborhood. The new reproduction probability is already lowered and taking away 36 of the previously 48 possible mates and mating location makes it even less likely for an organism to reproduce.

7.2.7 Removal of one species

When our Model is started **without any grass**, then the population still behaves the same as if grass was added at the beginning, because once a lion dies, it becomes grass. Therefore, the antelopes can potentially reach grass, survive and the balance is restored to the system. However, since the antelopes need to go past the lions (Lion Island: Section 7.2.2) to reach grass it is rather unlikely that lions and antelopes survive. The smaller the land is the smaller the probability that the land recovers from the initially strongly unbalanced situation. Figure 9 demonstrates the results of initially having no grass on the land and the antelopes unable to get to a grass patch generated by dead lions. Figure 10 shows the result of having no grass in a Lotka-Volterra model. In the Lotka-Volterra model removing all grass leads to the extinction of all organisms, because all antelopes starve and therefore all lions die of hunger too.

When **no antelopes** are added initially, all lions are guaranteed to starve, because they cannot reach any prey. Dying of hunger, the lions will become grass while already existing grass populates the entire land and hits a hard limit when there are no more cells to populate. Figure 11 shows the population graph of our simulation when no antelopes are present in the beginning. In a Lotka-Volterra model there is no maximum population the grass can reach. It will therefore continue growing to infinity. This is depicted in Figure 12. The trends between our simulation thus matches with the trend of the Lotka-Volterra model until the limit set by the map size is reached.

If initially **no lions** are added to the land there are two different outcomes: Either the antelopes eat all the grass and then starve leaving an empty land behind, or the antelopes allow some grass to grow by not overeating, resulting in both organisms to survive and continue living. The latter case is more probable in larger lands, whereas

in smaller lands the population seldom remains stable. Figure 13 shows the result of a simulation with no lions initially and the antelopes overfeeding whereas in Figure 14 the antelopes manage to survive by eating rationally. For the Lotka-Volterra model removing the lions simply reduces the model from a three species model to a two species model and thus, the population of the grass and of the antelopes could be self regulating and the curves might oscillate. However, because the Lotka-Volterra equations for three species require a different parameter setting to show an oscillating behavior, removing the lions and keeping the parameters set for three species results in the antelopes eating the all the grass and then slowly starving. This is shown in Figure 15.

Our results differs to the Lotka-Volterra model in some ways, because we consider locations and neighborhoods rather than purely basing our results on population count. Additionally the size of our land is limited to a fixed number of cells. Therefore a infinite population growth cannot occur in our model. The overall trends however match between the two models.

7.3 Overfeeding

As stated in the introduction we wanted to test our model by reproducing the disrespecting of the circle of life introduced by the main villain of the film "The Lion King". In our model this can be done by simply setting the MaxStomach attribute of the **lions** to ∞. As expected the lions eat every antelope and die out themselves because of starvation. The resulting graph is shown in Figure 17.
We decided to stretch this concept by making the **antelopes** very hungry instead of the lions. Our prediction was that the antelopes will eat all the grass, then die of either starvation or because the lions ate them. This will result in an overpopulation of the lions which will again lead to the starvation of the lions species. The results were only slightly surprising. The antelopes were unable to clear the land of grass instantly because the lions were able to eat a lot of antelopes in the process. This caused the population to oscillate a few times before the antelopes were able to eat all the grass. This result is dependent on the location of the organisms they are spawned in and on the direction in which they put their offspring. Our simulations all terminated after at most three oscillations (Shown in Figure 16).
Setting **both** the antelopes and the lions MaxStomach to ∞ surprisingly results in an oscillation. This can be explained by looking closer at the reproduction. A cell A can only reproduce once in every time-step, but can be used for reproduction by every other cell that can reach cell A. Therefore, with our 12-connected-neighborhood an organism can be used as a mating partner by up to 12 organisms. Mating, however, results in 1 point of stomach reduction. Having a stomach limit of only 1 will enable the organism to only produce one offspring. Having no limit of stomach and accessi-

ble food within the neighborhood means an organism can fill its stomach, reproduce with one of its neighbors and be a future potential mating partner for multiple other organisms. The reproduction rate is thus higher with MaxStomach equal to ∞ and antelopes reproduce quick enough to balance out the hungry lions.

Parameter	Value
start_X	600
start_Y	60
start_Z	60
a	0.2055
b	6.8493e-04
c	6.8493e-04
d	3.4247e-04
e	6.8493e-04
f	0.2055
g	6.8493e-04

Table 6: The choice of constant values for the comparison with the simulation results

8 Summary and Outlook

We have shown that using a cellular automaton we can closely reproduce a stable system that simulates the concept of the "Circle of life". We further conclude that our simulation resembles the Lotka-Volterra equations for the situation that all three species are present. Removing one species makes our results slightly differ from the Lotka-Volterra equation due to design choices such as lions becoming grass after death and having a map limit. The main difference however was that we have a location factor which the Lotka-Volterra does not take into account. All organisms are required in order to sustain on one land. Removing one organism causes the balance to be disturbed and usually means the extinction of another organism. Furthermore we realized that the size of the given land matters. Every organism needs space to grow and pass away, which can be achieved by creating a large land or by setting a small neighborhood to make the predators weaker. If the land is too small or the neighborhood is too large the cycle can break because one organism is able to take over. Not only the presence of each organism but the limits such as not overfeeding are vital for everyone's survival. We forced our organisms to respect each other by the use of parameters. If we allowed one organism to disrespect others by for example always feeding when its prey is nearby the balance is broken.

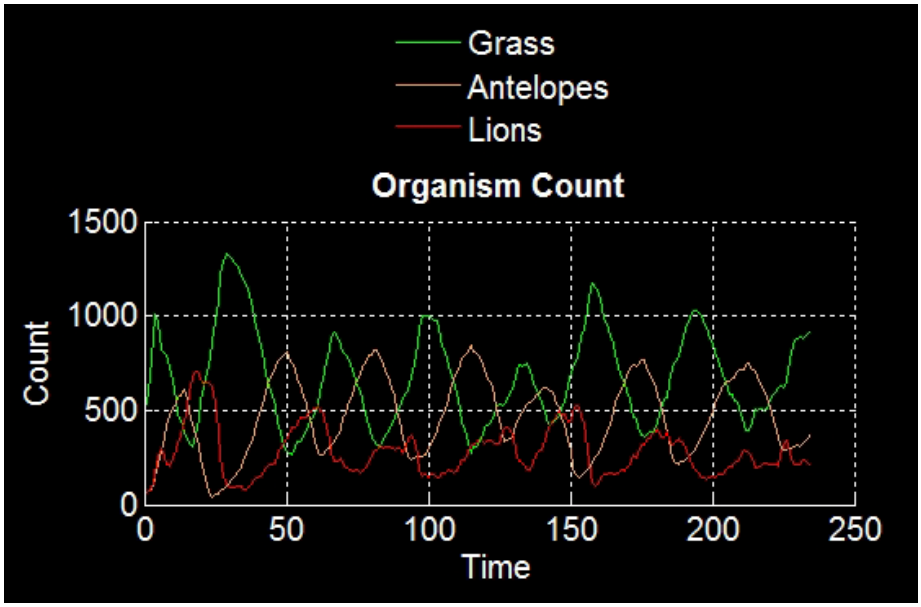


Figure 7: Circle of life simulation with oscillating behavior.

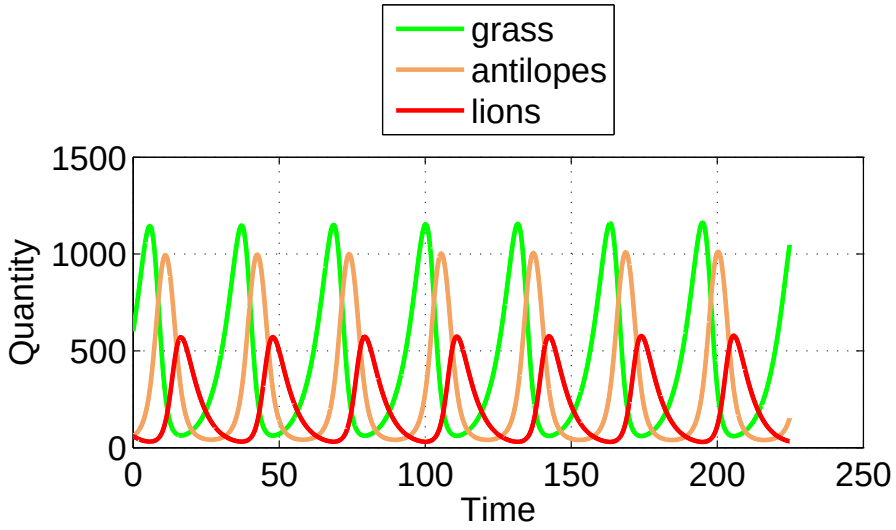


Figure 8: Lotka-Volterra equations for the parameters as shown in Table 6.

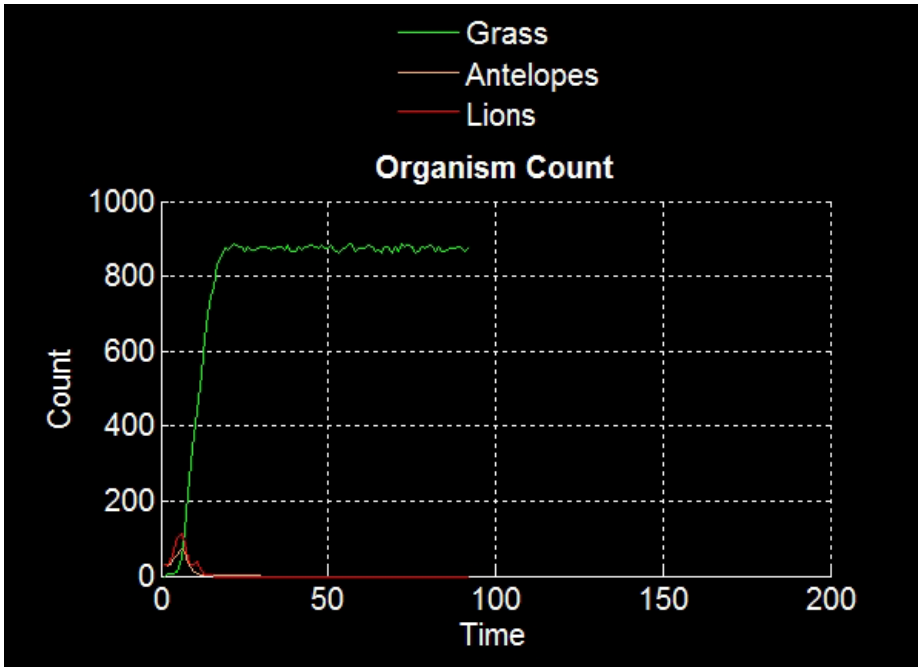


Figure 9: Circle of Life simulation with initially no grass. Dying lions generate grass

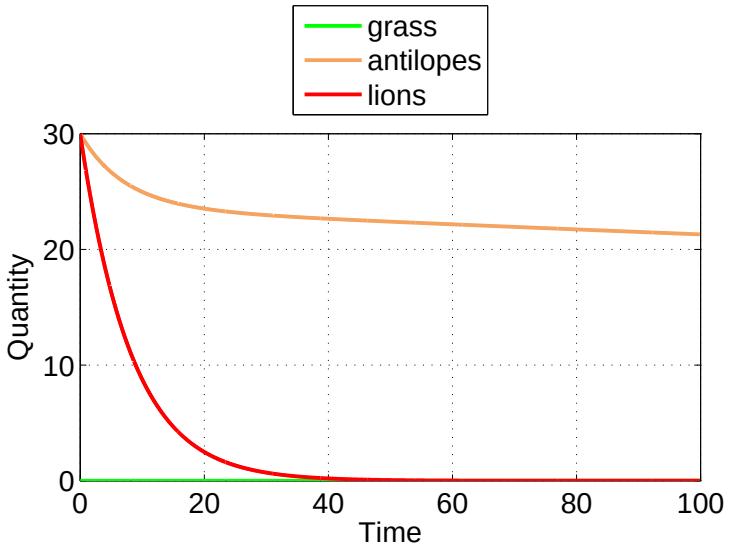


Figure 10: Lotka-Volterra equations, result of starting with zero grass.

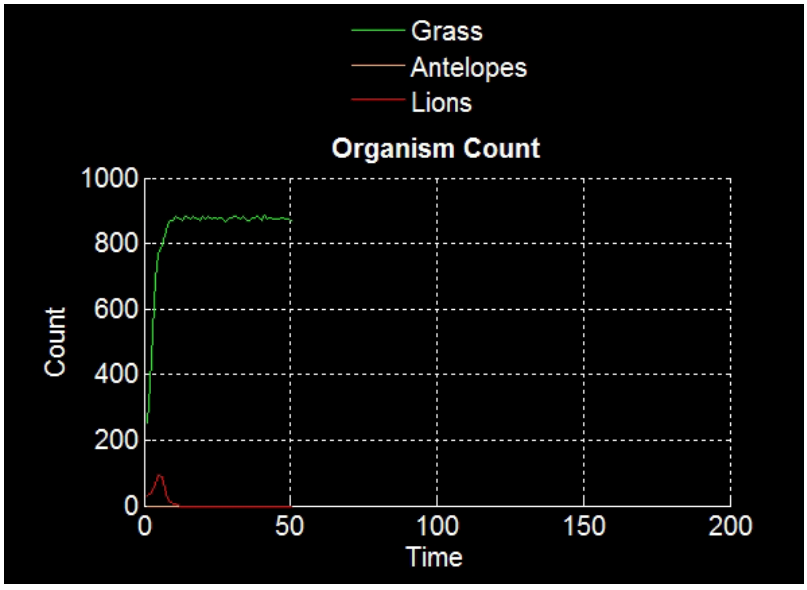


Figure 11: Circle of Life simulation with initially no antelopes.

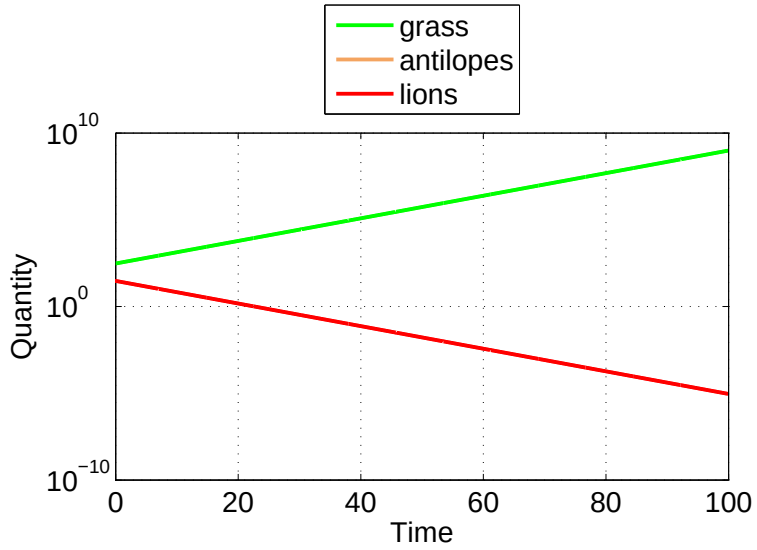


Figure 12: Lotka-Volterra equations, result of starting with zero antelopes.

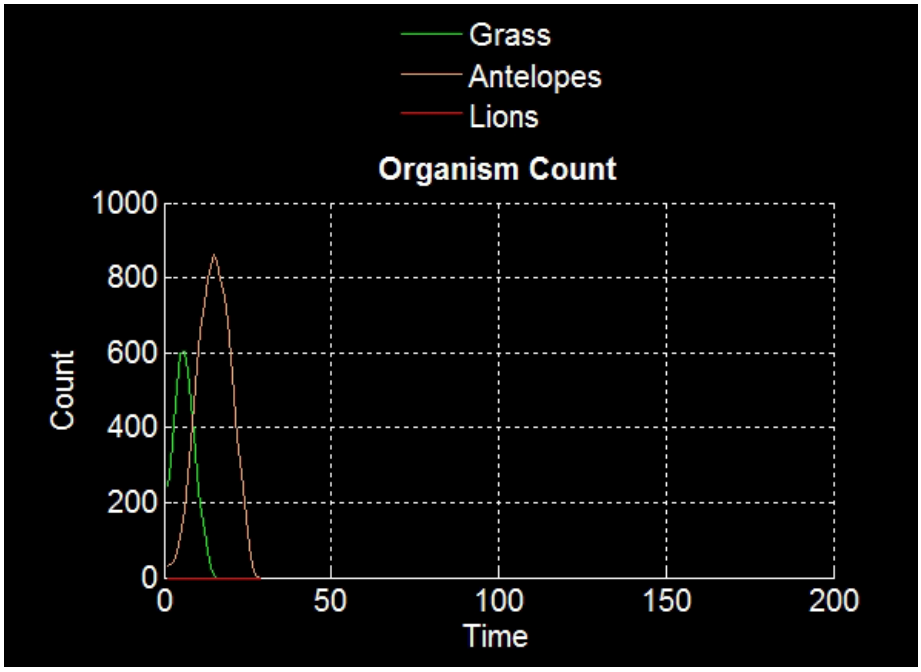


Figure 13: Circle of Life simulation with initially no lions where antelopes overfeed.

A Appendix

The title image was taken from [1].

B Code

B.1 Main file

```
% Simulate Circle Of life

%------CONSTANTS---------------------------------
SETUPINDEX = 1;
LAND_NUMBER = 3;

neigh_8 = getNeighbourhood(1); % Define the Moore neighborhood
neigh_8ext = [-1 -1; 0 -1; 1 -1; 1 0; 1 1; 0 1; -1 1; -1 0; 0 -2; 0 2; -2 ...
    0; 2 0];
neigh_2 = getNeighbourhood(2);
neigh_3 = getNeighbourhood(3);
neigh = neigh_8ext;
```

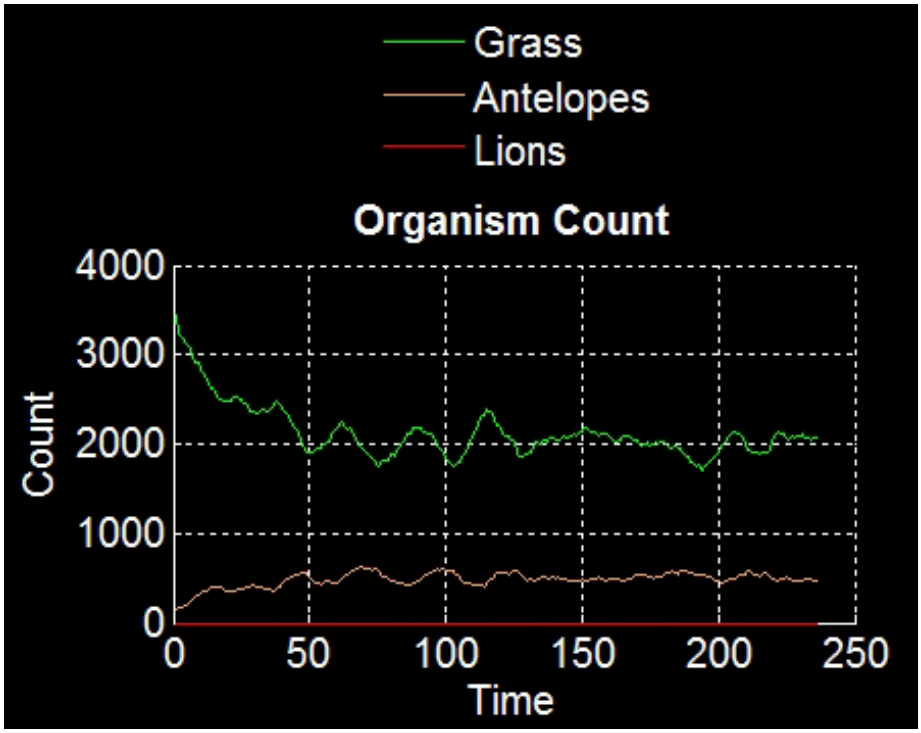


Figure 14: Circle of Life simulation with initially no lions where antelopes survive.

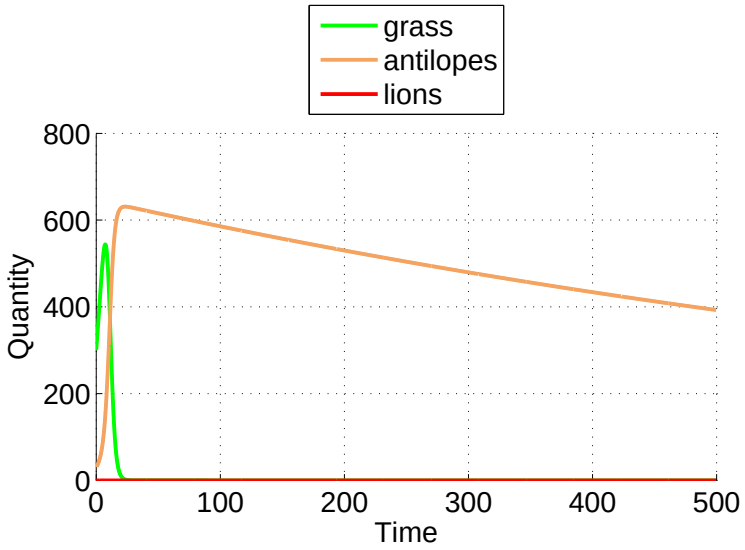


Figure 15: Lotka Volterra equations, result of starting with zero lions.

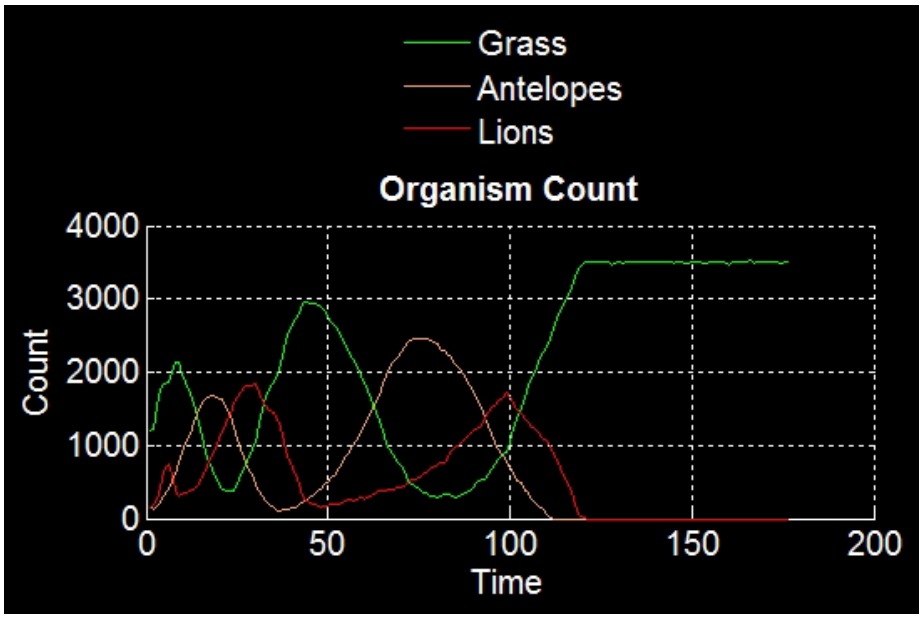


Figure 16: Simulation where antelopes overfeed.

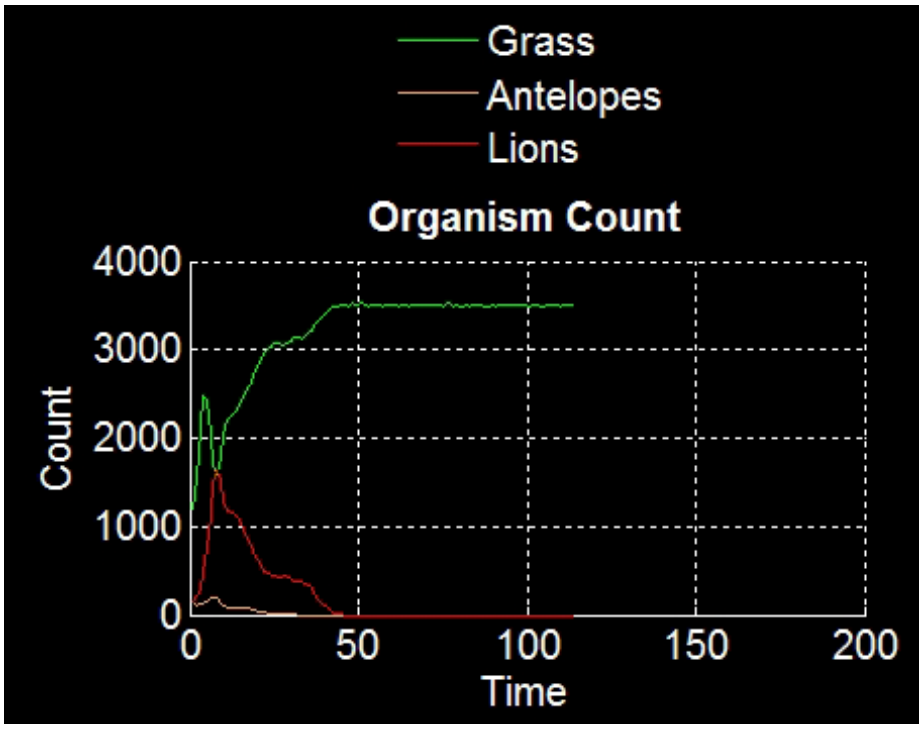


Figure 17: Simulation where lions overfeed.

```
% 0 = Nothing
% 1 = Grass
% 2 = Antelope
% 3 = Lion
NUMBER_OF_SPECIES = 3;
TIMESTEPS = 281474976710650; %Large number for nearly unlimited timesteps. ...
    Change window size to break timestep loop
NUMBER_OF_VARIABLES = 12;

typeInd = 1;
preyTypeInd = 2;    %The type of the prey
becomesInd = 3; %What type does this organism become after death

foodDigestInd = 4;   %Food digested in a day
stomachInd = 5; %Subtract "foodDigest" each day, +1 when eating, when equal ...
    to 0 organism becomes "becomesID"
maxStomachInd = 6;  %Organism will not feed if it's stomach reaches this ...
    constant

deathProbInd = 7;    %-1 after each turn, when reaching 0 organism becomes ...
    "becomeID"
fatnessInd = 8; %How many organisms can be fed by this organism
aliveInd = 9; %1 if alive, 0 if bitten
minStomachRepInd = 10; %minimum stomach required to reproduce;
repProbInd = 11;    %Probability of Reproduction
isOffspring = 12;    %0 = Cannot Reproduce?

NOTHING =   typeToOrganism(0,SETUPINDEX);
GRASS =     typeToOrganism(1,SETUPINDEX);
ANTELOPE =  typeToOrganism(2,SETUPINDEX);
LION =      typeToOrganism(3,SETUPINDEX);

%--------------------------------------------------
[X,Y,organismMat,organismCountMat] = ...
    getLand(LAND_NUMBER,NUMBER_OF_VARIABLES,SETUPINDEX);
deathCauseMat = zeros(3,3);

%Initialize Video Writer
fig = figure;
set(fig,'Color',[0 0 0])
set(fig, 'Position', [10 10 1300 800])
FILENAME = [datestr(clock, 30),'.avi'];
vidObj = VideoWriter(FILENAME);
vidObj.Quality= 100;
open(vidObj);

%Print initial Land
printLand(organismMat(:,:,1),organismCountMat(1,:),1,deathCauseMat);
```

```
disp('Initial Land Printed, 1 sec before start');
pause(1);

% generate random order for updates
xlocs = 1:X;
ylocs = zeros(1,X);
ylocs(1) = 1;
locations = [repmat(xlocs,[1,Y]) ; cumsum(repmat(ylocs,[1,Y]))];

% main loop, iterating the time variable, t
for t=1:TIMESTEPS
    allEmpty = 1; %If the land is empty we escape the loop
    deathlist = ones(X*Y,2);
    deathlistIndex = 0;

    %Adjust stomach -> vectorized for performance
    organismMat(:,:,stomachInd) = organismMat(:,:,stomachInd) - ...
        organismMat(:,:,foodDigestInd);
    randIndexes = randperm(X*Y);

    for iter=1:X*Y
        %Take a random location
        i = locations(1,randIndexes(iter));
        j = locations(2,randIndexes(iter));
        currentAnimal = organismMat(i,j,:);

        if (currentAnimal(aliveInd) == 0 || (currentAnimal(isOffspring) == 1))
            continue; %Animal is dead and is on deathlist or is born in ...
                this turn
        end

        if currentAnimal(typeInd) ~= NOTHING(typeInd)
            % Still Alive? (Age & Hunger)-----------------------------------
            allEmpty = 0;
            iDie = 0;

            if (currentAnimal(stomachInd) < 0)
                %Died of hunger
                iDie = 1;
                deathCauseMat(currentAnimal(typeInd),1) = ...
                    deathCauseMat(currentAnimal(typeInd),1) + 1;
            end

            if (currentAnimal(deathProbInd) > rand)
                %Died of Age
                iDie = 1;
                deathCauseMat(currentAnimal(typeInd),2) = ...
                    deathCauseMat(currentAnimal(typeInd),2) + 1;
            end
```

```
if (iDie == 1) %Kill organism
    newType = currentAnimal(becomesInd);
    newOrganism = typeToOrganism(newType,SETUPINDEX);
    organismMat(i,j,:) = newOrganism;
    continue;
end

%Check Neighborhood
%Organism
potentialMatingLocaction = [-1,-1];
potentialMate = [-1,-1];
randIndexes2 = randperm(size(neigh,1));

for k=1:size(neigh,1)
    i2 = mod((i+neigh(randIndexes2(k), 1))-1,X) + 1;
    j2 = mod((j+neigh(randIndexes2(k), 2))-1,Y) + 1;
    neighOrganism = organismMat(i2,j2,:);

    if (currentAnimal(typeInd) == 2)
        % disp('');
    end;

    switch neighOrganism(typeInd)

        case NOTHING(typeInd)   %Found Mating Loc
            potentialMatingLocaction = [i2,j2];

        case currentAnimal(preyTypeInd) %Found Food/Prey
            if (currentAnimal(stomachInd) < ...
                currentAnimal(maxStomachInd))
                if (neighOrganism(fatnessInd) > 0 && ...
                    neighOrganism(isOffspring) == 0)
                    organismMat(i,j,stomachInd) = ...
                        currentAnimal(stomachInd) + 1;
                    currentAnimal(stomachInd) = ...
                        currentAnimal(stomachInd) + 1;
                    organismMat(i2,j2,fatnessInd) = ...
                        neighOrganism(fatnessInd) - 1;

                    if (neighOrganism(aliveInd) == 1)   %Alive ...
                        or already dead?
                        organismMat(i2,j2,aliveInd) = 0;
                        deathlistIndex = deathlistIndex + 1;
                        deathlist(deathlistIndex,:) = [i2,j2];
                        deathCauseMat(neighOrganism(typeInd),3) ...
                            = ...
                            deathCauseMat(neighOrganism(typeInd),3) ...
                            + 1;
                    end
                end
```

```
            end
        case currentAnimal(typeInd) %Found Mate
            if (neighOrganism(aliveInd) == 1 &&...
                    neighOrganism(stomachInd) > ...
                        neighOrganism(minStomachRepInd) &&...
                    neighOrganism(isOffspring) == 0)
                %Is alive, has enough stomach and is not
                %offspring
                potentialMate = [i2,j2];
            end
    end

    %Check for mating location
    if neighOrganism(typeInd) == GRASS(typeInd)  %Found Grass ...
        Mating Loc
        if currentAnimal(typeInd) ~= GRASS(typeInd)
            if potentialMatingLocaction(1) == -1    %No Empty ...
                Location exists
                potentialMatingLocaction = [i2,j2];
            end
        end
    end
end


%Check if we can mate
if (potentialMate(1) ~= -1) && (potentialMatingLocaction(1) ~= -1)
    %Check our stomach is enough
    if currentAnimal(stomachInd) > currentAnimal(minStomachRepInd)
        if (rand < currentAnimal(repProbInd))
            %Reproduce
            currentMate = ...
                organismMat(potentialMate(1),potentialMate(2),:);
            a = potentialMatingLocaction(1);
            b = potentialMatingLocaction(2);

            newCurrentStomach = currentAnimal(stomachInd) - 1;
            newMateStomach = currentMate(stomachInd) - 1;

            organismMat(i,j,stomachInd) = newCurrentStomach;
            organismMat(potentialMate(1),potentialMate(2),stomachInd) ...
                = newMateStomach;

            organismMat(a,b,:) = ...
                typeToOrganism(currentAnimal(typeInd),SETUPINDEX);
            organismMat(a,b,stomachInd) = ((newCurrentStomach + ...
                newMateStomach)/2)+1;
        end
    end
end
```

```
    end
end


if allEmpty == 1
    break;
end

%remove dead organisms
%deathlist([1:deathlistIndex],:)
for i=1:deathlistIndex
    a = deathlist(i,1);
    b = deathlist(i,2);
    newOffspring = (organismMat(a,b,isOffspring) == 1);
    if (newOffspring)
        %Antelope eats Grass and 2 Lions/Antelopes put offspring on it.
        %We want to keep it this way
    else
        organismMat(a,b,:) = NOTHING; %becomes nothing after being eaten
    end
end

%Upgrade Offsprings
organismMat(:,:,isOffspring) =  zeros(X,Y);

%Set Counters
organismCounter(1) = sum(sum(organismMat(:,:,typeInd) == 1));
organismCounter(2) = sum(sum(organismMat(:,:,typeInd) == 2));
organismCounter(3) = sum(sum(organismMat(:,:,typeInd) == 3));
organismCountMat = [organismCountMat;organismCounter];

%{
%Log average stomachs:
stomachs = (organismMat(:,:,stomachInd));
logg('Stomach Annte: ', mean(stomachs((logical(organismMat(:,:,typeInd) ...
    == 2)))));
logg('Stomach Lion: ', mean(stomachs((logical(organismMat(:,:,typeInd) ...
    == 3)))));
%}

% Animate
printLand(organismMat(:,:,1),organismCountMat,t+1,deathCauseMat);
pause(0.00001);
%pause(0.2)

try
    writeVideo(vidObj,getframe(fig));
catch
    %Breaks if plot was resized
```

```
        break;
    end

end
close(vidObj);
winopen(FILENAME);
disp(['Finished: ',FILENAME]);
```

B.2 Organism Attribute Selection

This stores all attributes for each type. We have multiple combinations in order to quickly select them from the main class

```
function [ret] = typeToOrganism( type, setupIndex )

%{
typeInd = 1;
preyTypeInd = 2;    %The type of the prey
becomesInd = 3; %What type does this organism become after death

foodDigestInd = 4;   %Food digested in a day
stomachInd = 5; %Subtract "foodDigest" each day, +1 when eating, when equal ...
    to 0 organism becomes "becomesID"
maxStomachInd = 6;  %Organism will not feed if it's stomach reaches this ...
    constant

deathProbInd = 7;    %-1 after each turn, when reaching 0 organism becomes ...
    "becomeID"
fatnessInd = 8; %How many organisms can be fed by this organism
aliveInd = 9; %1 if alive, 0 if bitten
minStomachRepInd = 10; %minimum stomach required to reproduce;
repProbInd = 11;    %Probability of Reproduction

isOffspring = 12    %Can we reproduce?
%}

switch setupIndex %land 3 & neigh_8ext
    case 1
        switch type
            case 0      %1    2       3       4dige   5          6       ...
                7death                          8       9   10      11rep    12
                ret =   [0,   -1,     0,      0,      inf,      0,      ...
                    Inf,                              0,      0,  inf,    0, ...
                          1];
            case 1
                ret =   [1,   -1,     0,      0,      inf,      0,      ...
                    expectedAgeToDeathProb(5),     8,      1,  0,      0.7, ...
                         1];
            case 2
```

```
            ret =   [2,   1,      0,      1/2,    10,      10,       ...
                expectedAgeToDeathProb(10),     8,      1,  0,      1, ...
                     1];
        case 3
            ret =   [3,   2,      1,      1,    10,       10,      ...
                expectedAgeToDeathProb(10),     0,      1,  0,      1, ...
                    1];
    end
case 2 %Land 16 & neigh_8ext

    switch type
        case 0      %1    2       3       4dige   5          6       ...
           7death                          8       9   10      11rep    12
            ret =   [0,   -1,     0,      0,      inf,      0,      ...
                Inf,                              0,      0,  inf,    0, ...
                      1];
        case 1
            ret =   [1,   -1,     0,      0,      inf,      0,      ...
                expectedAgeToDeathProb(5),     8,      1,  0,      0.7, ...
                     1];
        case 2
            ret =   [2,   1,      0,      1/2,    10,      10,       ...
                expectedAgeToDeathProb(9),     8,      1,  0,      1, ...
                     1];
        case 3
            ret =   [3,   2,      1,      1/1,    10,       10,      ...
                expectedAgeToDeathProb(5),     0,      1,  0,      1, ...
                    1];
    end
case 3 %Land 18 & neigh_3

    switch type
        case 0      %1    2       3       4dige   5          6       ...
           7death                          8       9   10      11rep    12
            ret =   [0,   -1,     0,      0,      inf,      0,      ...
                Inf,                              0,      0,  inf,    0, ...
                      1];
        case 1
            ret =   [1,   -1,     0,      0,      inf,      0,      ...
                expectedAgeToDeathProb(5),     2,      1,  0,      0.7, ...
                     1];
        case 2
            ret =   [2,   1,      0,      1/5,    2,      2,       ...
                expectedAgeToDeathProb(10),     8,      1,  1,      1, ...
                     1];
        case 3
            ret =   [3,   2,      1,      1/3,    2,       2,      ...
                expectedAgeToDeathProb(10),     0,      1,  1,      ...
                0.6,     1];
    end
```

```
case 4 % Only antelopes and Grass Land 19

    switch type
        case 0      %1    2       3       4dige   5          6       ...
           7death                          8       9   10      11rep    12
            ret =   [0,   -1,     0,      0,      inf,      0,      ...
               Inf,                              0,      0,  inf,    0, ...
                     1];
        case 1
            ret =   [1,   -1,     0,      0,      inf,      0,      ...
               expectedAgeToDeathProb(5),     2,      1,  0,      0.7, ...
                    1];
        case 2
            ret =   [2,   1,      0,      1,    2,      2,       ...
               expectedAgeToDeathProb(10),     8,      1,  1,      1, ...
                    1];
        case 3
            ret =   [3,   2,      1,      1/3,    2,       2,      ...
               expectedAgeToDeathProb(10),     0,      1,  1,      ...
               0.6,     1];
    end

case 5 %Lions overfeeding Land 16 & neigh_8ext

   switch type
        case 0      %1    2       3       4dige   5          6       ...
           7death                          8       9   10      11rep    12
            ret =   [0,   -1,     0,      0,      inf,      0,      ...
               Inf,                              0,      0,  inf,    0, ...
                     1];
        case 1
            ret =   [1,   -1,     0,      0,      inf,      0,      ...
               expectedAgeToDeathProb(5),     8,      1,  0,      0.7, ...
                    1];
        case 2
            ret =   [2,   1,      0,      1/2,    10,      10,       ...
               expectedAgeToDeathProb(10),     12,      1,  0,      1, ...
                    1];
        case 3
            ret =   [3,   2,      1,      1/1,    10,       inf,      ...
               expectedAgeToDeathProb(10),     0,      1,  0,      1, ...
                   1];
    end
case 6 %Antilope overfeeding

    switch type
        case 0      %1    2       3       4dige   5          6       ...
           7death                          8       9   10      11rep    12
```

```
            ret =   [0,   -1,     0,      0,      inf,      0,      ...
                Inf,                                0,      0,  inf,    0, ...
                      1];
        case 1
            ret =   [1,   -1,     0,      0,      inf,      0,      ...
                expectedAgeToDeathProb(5),     8,      1,  0,      0.7, ...
                     1];
        case 2
            ret =   [2,   1,      0,      1/2,    10,       inf,       ...
                expectedAgeToDeathProb(10),     12,      1,  0,      1, ...
                     1];
        case 3
            ret =   [3,   2,      1,      1/1,    10,       13,       ...
                expectedAgeToDeathProb(10),     0,      1,  0,      1, ...
                    1];
    end
case 7 %Lions and Antilope overfeeding

    switch type
        case 0      %1    2       3       4dige   5          6       ...
           7death                           8       9   10      11rep    12
            ret =   [0,   -1,     0,      0,      inf,      0,      ...
                Inf,                                0,      0,  inf,    0, ...
                      1];
        case 1
            ret =   [1,   -1,     0,      0,      inf,      0,      ...
                expectedAgeToDeathProb(5),     8,      1,  0,      0.7, ...
                     1];
        case 2
            ret =   [2,   1,      0,      1/2,    10,       inf,       ...
                expectedAgeToDeathProb(10),     12,      1,  0,      1, ...
                     1];
        case 3
            ret =   [3,   2,      1,      1/1,    10,       inf,      ...
                expectedAgeToDeathProb(10),     0,      1,  0,      1, ...
                    1];
    end

case 8 %Map 18, neighbourhood 2

    switch type
        case 0      %1    2       3       4dige   5          6       ...
           7death                           8       9   10      11rep    12
            ret =   [0,   -1,     0,      0,      inf,      0,      ...
                Inf,                                0,      0,  inf,    0, ...
                      1];
        case 1
            ret =   [1,   -1,     0,      0,      inf,      0,      ...
                expectedAgeToDeathProb(5),     8,      1,  0,      0.7, ...
                     1];
```

```
            case 2
                ret =   [2,   1,      0,      1/2,    10,      10,       ...
                    expectedAgeToDeathProb(10),     8,      1,  0,      1, ...
                         1];
            case 3
                ret =   [3,   2,      1,      1/1,    10,       11,      ...
                    expectedAgeToDeathProb(10),     0,      1,  0,      1, ...
                        1];
        end

end
end
```

B.3 Expected Age to Death probability

This file is used to calculate the death probability given an expected life duration.

```
function [prob] = expectedAgeToDeathProb(age)
    prob = 1-(age/(age+1));
end
```

B.4 Generate Land

This file is used to create a new land. We have used a land index in order to select each quickly from the main class.

```
function [X,Y,organismMat, organismCounter] = ...
    getLand(landNumber,NUM_OF_VARIABLES, setupIndex)

NOTHING =   typeToOrganism(0, setupIndex);
GRASS =     typeToOrganism(1, setupIndex);
ANTELOPE =  typeToOrganism(2, setupIndex);
LION =      typeToOrganism(3, setupIndex);
organismCounter = zeros(1,3);
typeInd = 1;
isOffspring = 12;

switch landNumber
    case 0 %Only Antelope
        X=10;               % Grid size (XxY)
        Y =10;
        organismMat = createFlatLand(X,Y,NUM_OF_VARIABLES, setupIndex,2);
        organismMat(2,2,:) = ANTELOPE;
        organismMat(2,3,:) = ANTELOPE;
    case 1 %All Antelopes with Lions in middle
        X=50;               % Grid size (XxY)
```

```
    Y =50;
    organismMat = createFlatLand(X,Y,NUM_OF_VARIABLES, setupIndex,0);
    organismCounter(1) = 50*50;

    for i=1:X
        for j=1:Y
            if i >= 25 && i <= 30 &&...
                    j >= 25 && j <= 30
                organismMat(i,j,:) = LION;
                organismCounter(3) = organismCounter(3) + 1;
                organismCounter(1) = organismCounter(1) - 1;

            else
                organismCounter(2) = organismCounter(2) + 1;
                organismCounter(1) = organismCounter(1) - 1;
                organismMat(i,j,:) = ANTELOPE;
            end
        end
    end
case 2 %Manual Random
    X = 20;
    Y = 20;
    organismMat = createFlatLand(X,Y,NUM_OF_VARIABLES,setupIndex,0);

    organismMat(1,1,:) = GRASS;
    organismMat(10,10,:) = GRASS;
    organismMat(10,9,:) = GRASS;
    organismMat(8,8,:) = GRASS;
    organismMat(7,8,:) = GRASS;

    organismMat(2,3,:) = ANTELOPE;
    organismMat(5,5,:) = ANTELOPE;
    organismMat(2,4,:) = ANTELOPE;
    organismMat(5,4,:) = ANTELOPE;
    organismMat(14,15,:) = ANTELOPE;
    organismMat(15,17,:) = ANTELOPE;


    organismMat(6,5,:) = LION;
    organismMat(15,15,:) = LION;
    organismMat(6,6,:) = LION;
    organismMat(15,16,:) = LION;
case 3 %Randomized
    X = 40;
    Y = 40;
    organismMat = createFlatLand(X,Y,NUM_OF_VARIABLES,setupIndex,0);

    %random Grass
    rng('shuffle');
    for i=1:((X*Y)*(400/900))
```

```
        x =  randi([1 X]);
        y =  randi([1 Y]);
        disp(x);
        disp(y);
        organismMat(x,y,:) = GRASS;
    end
    %random Antelope
    rng('shuffle');
    for i=1:((X*Y)*(40/900))
        x =  randi([1 X]);
        y =  randi([1 Y]);
        disp(x);
        disp(y);
        organismMat(x,y,:) = ANTELOPE;
    end
    %random Lion
    rng('shuffle');
    for i=1:((X*Y)*(40/900))
        x =  randi([1 X]);
        y =  randi([1 Y]);
        disp(x);
        disp(y);
        organismMat(x,y,:) = LION;
    end
case 4  %Small Antelope Herd in Middle
    X=20;               % Grid size (XxY)
    Y =20;
    organismMat = createFlatLand(X,Y,NUM_OF_VARIABLES,setupIndex,0);

    for i=1:X
        for j=1:Y
            if i >= 8 && i <= 12 &&...
                    j >= 8 && j <= 12
                organismMat(i,j,:) = ANTELOPE;
            end
        end
    end
case 5 %Grass Land with herd of antelope in the middle
    X=20;               % Grid size (XxY)
    Y =20;
    organismMat = createFlatLand(X,Y,NUM_OF_VARIABLES,setupIndex,1);


    for i=1:X
        for j=1:Y
            if i > 8 && i < 12 &&...
                    j >= 8 && j <= 12
                organismMat(i,j,:) = ANTELOPE;
            end
        end
```

```
    end

case 6 %Only Antilipes
    X = 30;
    Y = 30;
    organismMat = createFlatLand(X,Y,NUM_OF_VARIABLES,setupIndex,2);
case 7 %Only Lions
    X = 10;
    Y = 10;
    organismMat = createFlatLand(X,Y,NUM_OF_VARIABLES,setupIndex,3);
case 8 %Only Grass + antelopes herd
    X = 50;
    Y = 50;
    organismMat = createFlatLand(X,Y,NUM_OF_VARIABLES,setupIndex,1);
    for i=1:X
        for j=1:Y
            if i >= 20 && i <= 30 &&...
                    j >= 20 && j <= 30
                organismMat(i,j,:) = ANTELOPE;
            end
        end
    end
case 9 %Half Antelope, Half Grass
    X = 50;
    Y = 50;
    organismMat = createFlatLand(X,Y,NUM_OF_VARIABLES,setupIndex,1);
    for i=X/2:X
        for j=1:Y

            organismMat(i,j,:) = ANTELOPE;
        end
    end
case 10 %All Antelopes  + stripe of grass
    X = 50;
    Y = 50;
    numStripes = 1;

    organismMat = createFlatLand(X,Y,NUM_OF_VARIABLES,setupIndex,2);
    for i=X/2:(X/2 + numStripes)
        for j=1:Y
            organismMat(i,j,:) = GRASS;
        end
    end

case 11 %All Grass + stripe of antelope
    X = 20;
    Y = 20;
    numStripes = 1;

    organismMat = createFlatLand(X,Y,NUM_OF_VARIABLES,setupIndex,1);
```

```
    organismCounter(1) = X*Y;
    for i=X/2:(X/2 + numStripes)
        for j=1:Y
            organismCounter(2) = organismCounter(2) +1;
            organismCounter(1) = organismCounter(1) -1;
            organismMat(i,j,:) = ANTELOPE;
        end
    end
case 12 %All Grass + stripe of antelope + stripe of lion
    X = 50;
    Y = 50;
    numAntilopStripes = 12;
    numLionStripes = 2;

    organismMat = createFlatLand(X,Y,NUM_OF_VARIABLES,setupIndex,1);
    organismCounter(1) = X*Y;
    for i=X/2:(X/2 + numAntilopStripes)
        for j=1:Y
            organismMat(i,j,:) = ANTELOPE;
            organismCounter(1) = organismCounter(1) -1;
            organismCounter(2) = organismCounter(2)+1;
        end
    end

    for i=(X/2+(numAntilopStripes/2)-(numLionStripes/2)) : ...
        (X/2+(numAntilopStripes/2) + (numLionStripes/2))
        for j=Y/2:Y
            organismMat(i,j,:) = LION;
            organismCounter(1) = organismCounter(1) -1;
            organismCounter(3) = organismCounter(3)+1;
        end
    end

case 13 %Random Grass
    X = 10;
    Y = 10;
    organismMat = createFlatLand(X,Y,NUM_OF_VARIABLES,setupIndex,0);

    %fill half with random Grass
    rng('shuffle');
    for i=1:(X*Y)/2
        x =  randi([1 X]);
        y =  randi([1 Y]);
        while (organismMat(x,y,typeInd) == GRASS(typeInd))
            x =  randi([1 X]);
            y =  randi([1 Y]);
        end
        organismMat(x,y,:) = GRASS;
    end
    organismCounter(1) = X*Y/2;
```

```
case 14 %Box of lion in box of antelope in box of grass
    X = 6;
    Y = 6;
    organismMat = createFlatLand(X,Y,NUM_OF_VARIABLES,setupIndex,2);

    for i=1:X
        for j=1:Y
            if (i >= 2 && i <= 5) && (j >= 2 && j <= 5)
                organismMat(i,j,:) = GRASS;
            end


            if (i == 3 || i == 4) && (j == 3 || j == 4)
                organismMat(i,j,:) = LION;
            end
        end
    end
case 15 %All random
    X = 30;
    Y = 30;
    organismMat = createFlatLand(X,Y,NUM_OF_VARIABLES,setupIndex,0);

    %fill half with random Grass
    rng('default');

    for i=1:(X*Y)/4
        x =  randi([1 X]);
        y =  randi([1 Y]);
        while (organismMat(x,y,typeInd) == LION(typeInd))
            x =  randi([1 X]);
            y =  randi([1 Y]);
        end
        organismMat(x,y,:) = LION;
    end
    for i=1:(X*Y)/2
        x =  randi([1 X]);
        y =  randi([1 Y]);
        while (organismMat(x,y,typeInd) == GRASS(typeInd))
            x =  randi([1 X]);
            y =  randi([1 Y]);
        end
        organismMat(x,y,:) = GRASS;
    end
    for i=1:(X*Y)/2
        x =  randi([1 X]);
        y =  randi([1 Y]);
        while (organismMat(x,y,typeInd) == ANTELOPE(typeInd))
            x =  randi([1 X]);
            y =  randi([1 Y]);
```

```
        end
        organismMat(x,y,:) = ANTELOPE;
    end
case 16 %Randomized
    X = 60;
    Y = 60;
    organismMat = createFlatLand(X,Y,NUM_OF_VARIABLES,setupIndex,0);

    %random Grass
    rng('shuffle');
    for i=1:((X*Y)*(400/900))
        x =  randi([1 X]);
        y =  randi([1 Y]);
        disp(x);
        disp(y);
        organismMat(x,y,:) = GRASS;
    end
    %random Antelope
    rng('shuffle');
    for i=1:((X*Y)*(40/900))
        x =  randi([1 X]);
        y =  randi([1 Y]);
        disp(x);
        disp(y);
        organismMat(x,y,:) = ANTELOPE;
    end
    %random Lion
    rng('shuffle');
    for i=1:((X*Y)*(40/900))
        x =  randi([1 X]);
        y =  randi([1 Y]);
        disp(x);
        disp(y);
        organismMat(x,y,:) = LION;
    end
case 17 %Randomized big
    X = 100;
    Y = 100;
    organismMat = createFlatLand(X,Y,NUM_OF_VARIABLES,setupIndex,0);

    %random Grass
    rng('shuffle');
    for i=1:((X*Y)*(400/900))
        x =  randi([1 X]);
        y =  randi([1 Y]);
        organismMat(x,y,:) = GRASS;
    end
    disp('added grass');
    %random Antelope
    rng('shuffle');
```

```
    for i=1:((X*Y)*(40/900))
        x =  randi([1 X]);
        y =  randi([1 Y]);
        organismMat(x,y,:) = ANTELOPE;
    end
    disp('added antelope');
    %random Lion
    rng('shuffle');
    for i=1:((X*Y)*(10/900))
        x =  randi([1 X]);
        y =  randi([1 Y]);
        organismMat(x,y,:) = LION;
    end
    disp('added Lions');

case 18 %Chasing pattern forced
    X = 60;
    Y = 60;
    organismMat = createFlatLand(X,Y,NUM_OF_VARIABLES,setupIndex,1);

    %Antelope
    for i=20:30
        for j=25:35
            organismMat(i,j,:) = ANTELOPE;
        end;
    end
    %Lion
    for i=20:25
        for j=25:35
            organismMat(i,j,:) = LION;
        end;
    end
    for i=20:30
        for j=36:36
            organismMat(i,j,:) = LION;
        end;
    end
case 19 %Grass + Random Antelopes
    X = 60;
    Y = 60;
    organismMat = createFlatLand(X,Y,NUM_OF_VARIABLES,setupIndex,1);

    %random Antelope
    rng('shuffle');
    for i=1:((X*Y)*(40/900))
        x =  randi([1 X]);
        y =  randi([1 Y]);
        organismMat(x,y,:) = ANTELOPE;
    end
case 20 %Chasing pattern forced
```

```
        X = 40;
        Y = 40;
        organismMat = createFlatLand(X,Y,NUM_OF_VARIABLES,setupIndex,1);

        %Antelope
        for i=20:30
            for j=25:35
                organismMat(i,j,:) = ANTELOPE;
            end;
        end
        %Lion
        for i=20:25
            for j=25:35
                organismMat(i,j,:) = LION;
            end;
        end
        for i=20:30
            for j=36:36
                organismMat(i,j,:) = LION;
            end;
        end
end

%No one isOffpsring
organismMat(:,:,isOffspring) =  zeros(X,Y);

%Set Counters
organismCounter(1) = sum(sum(organismMat(:,:,typeInd) == 1));
organismCounter(2) = sum(sum(organismMat(:,:,typeInd) == 2));
organismCounter(3) = sum(sum(organismMat(:,:,typeInd) == 3));
```

B.5 Generate Flatland

This is used by the land generator. It outputs an X by Y matrix and fills it with the given type of organism.

```
function [organismMat] = ...
    createFlatLand(X,Y,NUMBER_OF_VARIABLES,setupIndex,LandType)
%   CREATEFLATLAND Creates (X,Y,NUMBER_OF_VARIABLES)-matrix populated with
%   LandType-organism
% 1. get Organism type
% 2. transpose in 3D to get a (1,1,NUMBER_OF_VARIABLES)-vector
% 3. replicate this vector X*Y times to fill the entire land with this
%    organism
organismMat = ...
    repmat(permute(typeToOrganism(LandType,setupIndex),[1,3,2]),[X,Y,1]);
end
```

B.6 Define Neighborhood

This is used to generate the neighborhood and is used in the main file.

```
function [ neigh] = getNeighbourhood(s)
%GETNEIGHBOURHOOD Generates a von Neumann neighborhood with manhattan ...
    disctance s
xDir = -s:s;
yDir = ones(1,(s*2)+1);
yDir(1) = 1;
neigh = [];
for neighIt = -s:s
    if (neighIt ~= 0)
        addNeigh = [xDir', ones((s*2)+1,1)*neighIt];
    else
        addNeigh = [[-s:-1,1:s]', ones((s*2),1)*neighIt];
    end
    neigh = [neigh; addNeigh];
end
end
```

B.7 Print Land

Used for printing plots

```
function [] = printLand( mat, organismCountMat, t, deathCauseMat)
%PRINTLAND Summary of this function goes here
%   Detailed explanation goes here

    color_Lion = [1 0 0];
    color_Antelope = [244/255, 164/255, 96/255];
    color_Grass = [0 1 0];
    color_Nothing = [0 0 0];
    FONTSIZE = 15;
    clf;                                % Clear figure


    %Land-----------------------------------------------
    subplot(2, 2, 1);
    imagesc(mat, [0 3]);                  % Display grid
    colormap([color_Nothing; color_Grass; color_Antelope; color_Lion]);
    set(gca,'YTickLabel',[]);
    set(gca,'XTickLabel',[]);
    axis on;

    %Title
    titl = title('Land','fontWeight','bold');
```

```
set(titl,'FontSize',FONTSIZE);
set(titl,'Color',[1,1,1]);

%Axis
set(gca, 'XColor', 'w');
set(gca, 'YColor', 'w');

%Legend
set(legend,'TextColor',[1,1,1]);
set(gca,'FontSize',FONTSIZE);



%DeathCause Age/Hunger Death-----------------------
%1: Hunger
%2: Age
%3: Eaten
subplot(2, 2, 2);
barPlot = bar(deathCauseMat,'stack');
XLABELS = {'Grass','Antelope','Lion'};
set(gca,'Color',[0 0 0]);
set(gca,'FontSize',FONTSIZE);
set(barPlot,{'FaceColor'},{[1,204/255,0];[0,153/255,1];[1,0,0]});
grid on;

%Title
titl = title('Death Causes','fontWeight','bold');
set(titl,'FontSize',FONTSIZE);
set(titl,'Color',[1,1,1]);


%Axis
xlab = xlabel('(Organism)');
ylab = ylabel('Total Deaths');
set(xlab,'FontSize',FONTSIZE);
set(ylab,'FontSize',FONTSIZE);
set(gca, 'XColor', 'w');
set(gca, 'YColor', 'w');
set(gca,'XTickLabel',XLABELS);
set(gca,'FontSize',FONTSIZE);

%Legend
legend({'Hunger', 'Age', 'Eaten'});
set(legend,'Color',[0,0,0]);
set(legend,'TextColor',[1,1,1]);
set(legend,'Location','EastOutside');

%DeathCause (%) Age/Hunger Death-----------------------
%1: Hunger
%2: Age
```

```
%3: Eaten
x = deathCauseMat./repmat(sum(deathCauseMat,2),1,3);
x = x*100;
subplot(2, 2, 4);
barPlot = bar(x,'stack');
XLABELS = {'Grass','Antelope','Lion'};
set(gca,'Color',[0,0,0]);
set(gca,'FontSize',FONTSIZE);
set(barPlot,{'FaceColor'},{[1,204/255,0];[0,153/255,1];[1,0,0]});
ylim([0 100]);
grid on;

%Title
titl = title('Death Causes in Percentage','fontWeight','bold');
set(titl,'FontSize',FONTSIZE);
set(titl,'Color',[1,1,1]);

%Axis
xlab =xlabel('(Organism)');
ylab =ylabel('Deaths in Percentage (%)');
set(xlab,'FontSize',FONTSIZE);
set(ylab,'FontSize',FONTSIZE);
set(gca, 'XColor', 'w');
set(gca, 'YColor', 'w');
set(gca,'XTickLabel',XLABELS)

%Legend
legend({'Hunger', 'Age', 'Eaten'});
set(legend,'Color',[0,0,0]);
set(legend,'TextColor',[1,1,1]);
set(legend,'Location','EastOutside');

%Counter--------------------------------
subplot(2, 2, 3);
x = 1:t;
hold on;
l1 = plot(x,organismCountMat(:,1),'DisplayName','Grass');
l2 = plot(x,organismCountMat(:,2),'DisplayName','Antelopes');
l3 = plot(x,organismCountMat(:,3),'DisplayName','Lions');
set(l1,'Color',color_Grass);
set(l2,'Color',color_Antelope);
set(l3,'Color',color_Lion);
set(gca,'Color',[0,0,0]);
set(gca,'FontSize',FONTSIZE);
if (t < 200)
    xlim([0 200]);
else
    xlim('auto');
end
grid on;
```

```
    %Title
    titl =title('Organism Count','fontWeight','bold');
    set(titl,'FontSize',FONTSIZE);
    set(titl,'Color',[1,1,1]);

    %Axis
    xlab = xlabel('Time');
    ylab = ylabel('Count');
    set(xlab,'FontSize',FONTSIZE);
    set(ylab,'FontSize',FONTSIZE);
    set(gca, 'XColor', 'w');
    set(gca, 'YColor', 'w');

    %Legend
    legend(gca,'show');
    set(legend,'Color',[0,0,0]);
    set(legend,'TextColor',[1,1,1]);
    set(legend,'Location','NorthOutside');

    hold off;
end
```

B.8 Lotka-Volterra

This is used for plotting a reference Lotka-Volterra equation.

```
function [] = LotkaVolterraThree()

clf;
color_Lion = [1 0 0];
color_Antelope = [244/255, 164/255, 96/255];
color_Grass = [0 1 0];
FONTSIZE = 15;

X_START = 600;
WAVELENGTH = 73;
ENDTIME = 225;

% timestep
dt=0.00006*WAVELENGTH;
% iterations
iter=ceil(333*ENDTIME*(50/WAVELENGTH));

% initialize vectors
x=zeros(iter,1); y=zeros(iter,1); z=zeros(iter,1); t=zeros(iter,1);
l = 0.05/WAVELENGTH;
```

```
x(1)=1*X_START; y(1)=0.1*X_START; z(1)=0.1*X_START; t(1)=0; % starting point

a=0.5*X_START*l; b=1*l; c=1*l; d=0.5*l; e=1*l; f=0.5*X_START*l; g=1*l;

% all ones
%x(1) = 0.5;y(1) = 1; z(1) =2;
%a =1;b=1;c=1;d=1;e=1;f=1;g=1;

for i=2:iter
    dx=a*x(i-1)- b*x(i-1)*y(i-1);
    dy=-c*y(i-1) + d*x(i-1)*y(i-1) - e*y(i-1)*z(i-1);
    dz=-f*z(i-1) + g*y(i-1)*z(i-1);
    x(i)=x(i-1)+dt*dx;
    y(i)=y(i-1)+dt*dy;
    z(i)=z(i-1)+dt*dz;
    t(i)=t(i-1)+dt;
end

plot(t,x,'LineWidth',2,'Color',color_Grass);
hold on;
plot(t,y,'LineWidth',2,'Color',color_Antelope);
plot(t,z,'LineWidth',2,'Color',color_Lion);

%Legend
legend('grass','antilopes','lions');
legend(gca,'show');
set(legend,'Location','NorthOutside');


%Axis
xlab = xlabel('Time');
ylab = ylabel('Quantity');
set(xlab,'FontSize',FONTSIZE);
set(ylab,'FontSize',FONTSIZE);
set(gca, 'XColor', 'w');
set(gca, 'YColor', 'w');

set(gca,'XColor',[0,0,0]);
set(gca,'YColor',[0,0,0]);
set(gcf,'Color',[1,1,1]);
set(gca,'FontSize',FONTSIZE);
grid on;
hold off;

end
```

References

[1] Image Reincarnation. http://melissaj007.blogspot.ch/2011/04/reincarnation.html, May 2014.

[2] A.A. Berryman. The Origins and Evolution of Predator-Prey Theory. *Ecology*, page 15301535, 1992.

[3] Erica Chauvet et al. A Lotka-Volterra Three-species Food Chain. pages 243–255, 2002.

[4] Dr. F. Antwerpes G. Römer. Caenorhabditis elegans. http://flexikon.doccheck.com/de/Caenorhabditis_elegans, 5 2014.

[5] Martin Gardner. Mathematical Games The fantastic combinations of John Conway's new solitaire game "life". *Scientific American 223*, pages 120–123, Oct 1970.

[6] A.J. Lotka. Elements of Physical Biology, Williams and Wilkins. 1925.

[7] R. Allers R. Minkoff. The lion king. http://www.imdb.com/title/tt0110357/, 1994.

[8] V. Volterra. Variazioni e fluttuazioni del numero dindividui in specie animali conviventi. *Mem. Acad. Lincei Roma*, pages 31–113, 1926.

[9] Eric W. Weisstein. Moore neighborhood. From MathWorld–A Wolfram Web Resource. http://mathworld.wolfram.com/MooreNeighborhood.html, 5 2014.

[10] Eric W. Weisstein. von neumann neighborhood. From MathWorld–A Wolfram Web Resource. http://mathworld.wolfram.com/vonNeumannNeighborhood.html, 5 2014.