CSE/ECE 6730 Project 2: Population Dynamics Using the Cellular Automata Model

Team

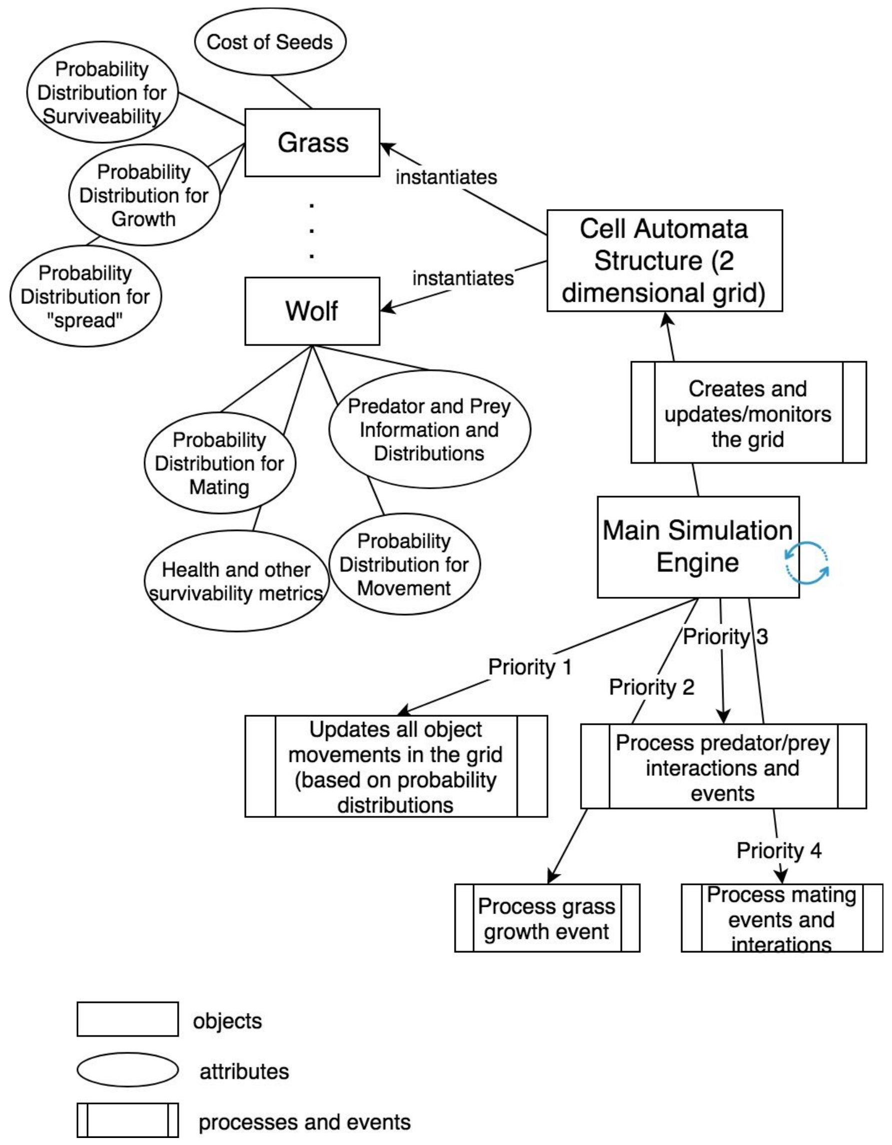
- Matthew Schieber, mschieber3
- Jordi Wolfson-Pou, jwp3
- Abhay Dalmia, adalmia3

Project Description

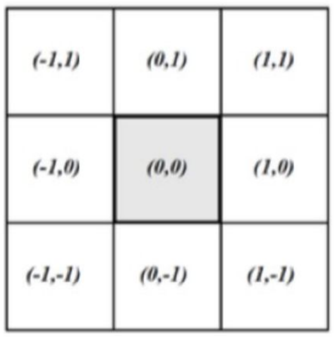
This project attempts to simulate the behavior of animal populations to determine which species of grass seed to plant a field with and whether the populations are sustainable. The customer of the simulation is a conservation agency that plans to maintain a field of prey (rabbits) and predators (coyotes and wolves). The agency knows that the sustainability of the populations is highly dependent on which grass seed they plant the field with. Each grass species grows at a different rate and will directly affect the rabbit population. For instance, if a grass species grows rapidly, then that grass is more likely to sustain a higher population of rabbits. The grass seeds also vary in cost, so the agency needs to spend just enough to achieve the desired population sustenance. Moreover, both the coyotes and wolves will be competing for hunting rabbits. Wolves are more dominant predators, and coyotes will usually avoid wolves if possible [1]. Since wolves and coyotes compete for the same resource, wolves will chase away or even kill coyotes. To survive, the coyotes will often migrate away and in between wolf populations [2]. Even so, the agency is hoping that with a large enough field, the coyotes will be capable of surviving while competing with the wolves. We will develop a simulator using the cellular automata model in order to predict the sustainability of these animal populations.

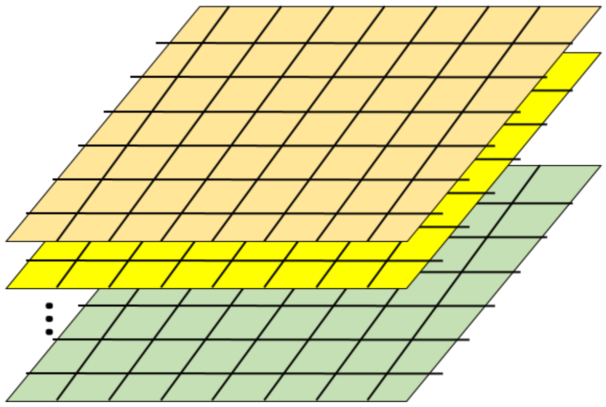
Conceptual Model

The diagram below shows an object-attribute-events structure of the software that we will develop. It presents the main events that we will handle, and the objects attributes that will be mutated and changed through this simulation.


Cost of Seeds
Probability Distribution for Surviveability
Probability Distribution for Growth
Probability Distribution for "spread"
Grass
instantiates
Cell Automata Structure (2 dimensional grid)
instantiates
Wolf
Creates and updates/monitors the grid
Probability Distribution for Mating
Predator and Prey Information and Distributions
Health and other survivability metrics
Probability Distribution for Movement
Main Simulation Engine
Priority 1
Priority 2
Priority 3
Updates all object movements in the grid (based on probability distributions
Process predator/prey interactions and events
Priority 4
Process grass growth event
Process mating events and interations
objects
attributes
processes and events

The simulation engine will first initialize the cell automata grid. We employed a multi-dimensional CA using a von Neumann neighborhood (shown below). Each grid spot on this 2D array is a list of objects that are currently in that grid location. Objects can be any animal (Wolves, Coyotes, Rabbits), or Vegetation (Grass). An example of each is shown below with attributes that will be used in deciding events or will be mutated as a result of events. For example, the wolf's probability distribution of mating controls how likely a wolf will produce offspring. Other metrics define the current health of a wolf, and the amount of health the wolf loses every time period that it goes without eating food/prey among other attributes. If a wolf produces offspring, the offspring inherits the health of the parent. Movement of wolves (and other similar objects) will also be controlled by some version of probability distribution based on where the "food" is on the grid and if the animal has to avoid predators around them in other cells. In other words, animals will look at neighboring grid cells, and, based on what occupies them, will compute a probability distribution (see below).





Von Neumann neighborhood (right) and multi-dimensional CA visualization (left)

For vegetation (like grass), the attributes will be slightly different. The attributes in this case will control the probability that grass growing in one grid location will spread seeds to adjacent grid locations, and the growth rate of the grass itself (again modeled with some distribution). The probability of spreading to to a new grid location will follow an exponential distribution. More expensive seeds will have a lower "index" *i*.

Once the simulation engine creates a grid based on these objects (where number of each object and object placement can either be random or user input), for each time slice, it will process 4 basic events. It will first update the object movements to the new locations. Then, for the new positions, it will process grass growth events first, then prey/predator interactions, and finally mating events and interactions. This will continue on for a certain number of iterations and relevant data will be collected.

Assumptions

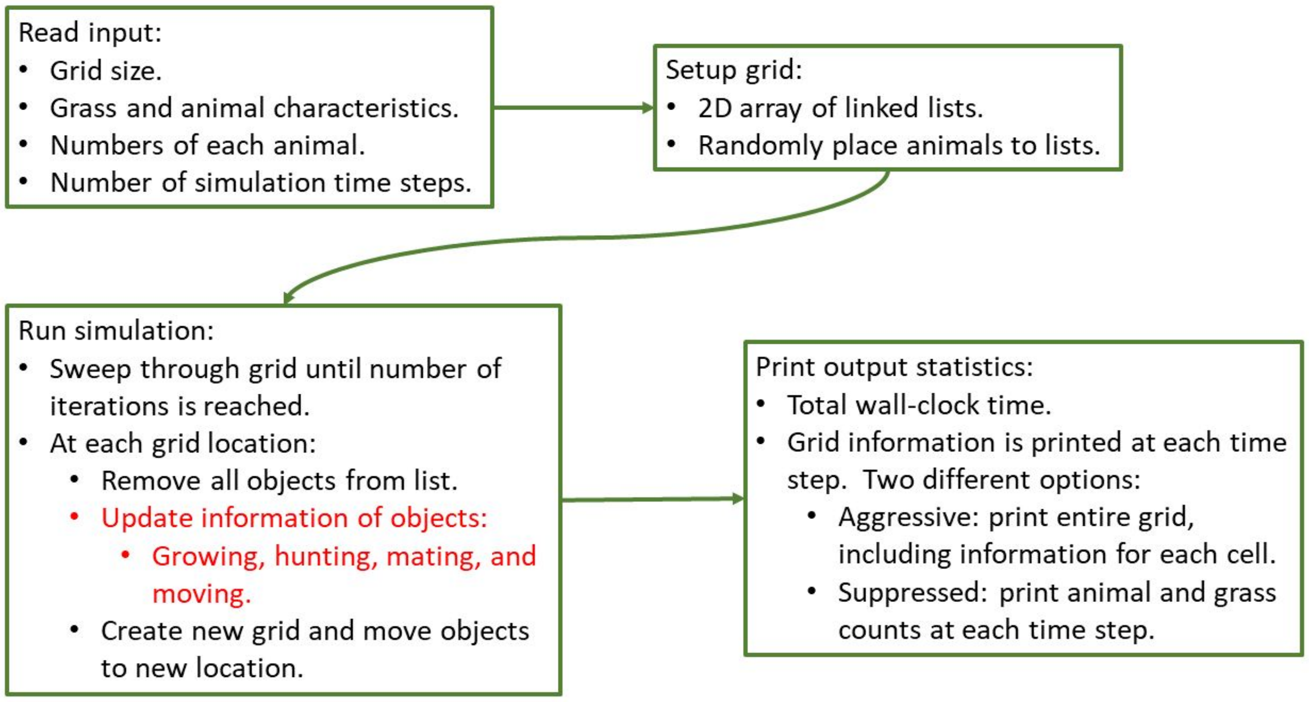
- Although rabbits typically require several types of vegetation [3], we will assume they only need to consume one grass species.
- We will assume that the grass will get enough rain to sustain its growth rate, ignoring weather conditions such as droughts and floods.

Software Components

The diagram with the conceptual model above is a good description of the major software components to be developed. Each entity (wolf and grass as an example) and their attributes will be developed as separate objects under an object-oriented design approach. The grid will also be an object with a 2-D array, and each position will be a list of objects that currently exists in that position. Each of these objects have level abstraction from each other and can be developed independently.

The main simulation engine will instantiate the grid and the objects that the grid will have initially. The simulation engine will then execute the 4 events mentioned in the conceptual model in the priority as stated above. These events are the other major components of the software and can also be worked on independently, as the object design has a set interface of attributes and functions that these events can access, call and modify to achieve the stated goal of each event.

The overall program flow is shown in the figure below. The main components are (1) reading the user input, (2) setting up the grid, (3) the main simulation loop (the red indicates what the majority of the code does), and (4) printing the output. The code is written in Python.


Read input:
• Grid size.
• Grass and animal characteristics.
• Numbers of each animal.
• Number of simulation time steps.
Setup grid:
• 2D array of linked lists.
• Randomly place animals to lists.
Run simulation:
• Sweep through grid until number of iterations is reached.
• At each grid location:
• Remove all objects from list.
• Update information of objects:
• Growing, hunting, mating, and moving.
• Create new grid and move objects to new location.
Print output statistics:
• Total wall-clock time.
• Grid information is printed at each time step. Two different options:
• Aggressive: print entire grid, including information for each cell.
• Suppressed: print animal and grass counts at each time step.

Files
Our software is contained in the file field.py. A README is provided that instructs the user how to run the software.

Parameters
Here is a list of parameters that the user can alter, which can be done by opening the main (and only) file field.py:

- The initial number of rabbits, coyotes, and wolves.
- The number of days it takes for each animal to starve: the total health of each animal is two times the number of days it takes for the animal to starve. In other words, each animal has a grace period in which it is healthy and not hungry.
- The rate at which each animal breeds, i.e., the probability of an animal breeding at each time step.
- The rate at which reproduction can occur, e.g., the user can restrict mating to happen only every 6 time steps.
- The probability of a coyote or wolf successfully killing another animal. Wolves hunt coyotes and rabbits, while coyotes only hunt rabbits.
- The number of simulation time steps.
- The size of the field.

- The movement distribution of each animal. Each animal has a probability of moving to a neighboring cell.
- The type of grass seed being used. This is a tuple. The first element is how much the grass grows, and the second element is how often it grows.

Grid Setup

The grid is initialized with ungrown grass at each cell. Each cell is a dictionary of lists, where the keys are animals. The animals are placed in random cells, and start with full health. For our final simulation, we plan on grouping like animals together in order to make the simulation more realistic (e.g. herds).

Main Simulation Loop

Each time step in the main loop involves sweeping through and updating each grid cell. Grid cells can be updated concurrently. If an event is based on some probability, we generate a uniform random number in the range [0,1]. Success happens if the random number is less than the probability. Updating the grid at each time step is broken up into four parts.:

- **Growth:** the grass in each cell grows if the simulation is at a time step in which grass is allowed to grow (see **Parameters**).
- **Animal interactions:** each animal interacts with some other animals or grass. The interactions are handled in sequence as follows.
    - The rabbits first interact with the grass. If a rabbit is hungry, it eats some of the grass in the cell, which decreases the amount of grass in that cell. If there is no grass in that cell and a rabbit has a health value of one, the rabbit health decrements to zero. Consequently the rabbit dies and is removed from the simulation.
    - The coyotes interact with the rabbits in the same way that the rabbits interact with the grass. The key difference is that coyotes have to hunt the rabbits, which is based on a probability. If a rabbit is successfully killed, the coyote's health is replenished to full, and the coyote is no longer hungry and stops hunting. Coyotes die in the same way rabbits die.
    - Wolves hunt both rabbits and coyotes. The key difference between wolves and coyotes hunting is that wolves hunt coyotes even if they aren't hungry. This simulates the territorial nature of wolves. If a wolf happens to be hungry when successfully killing a coyote, then the wolf's health is replenished. Wolves die in the same way as the rest of the animals.
- **Animal mating:** if mating is allowed at the current time step (see **Parameters**), each animal has a probability of mating (again see **Parameters**). If an animal successfully produces offspring, the offspring is initialized to have the same health as the parent.
- **Movement:** this function handles the animal movement to neighboring cells (the animal may also stay put). A new empty grid is created with grass at each cell. The health of the grass is taken from the previous grid. The animals in the previous grid then move to neighboring cells on the new grid based on the movement probability distribution (see **Parameters).** There is a starting probability of moving based on distance, i.e., cells in diagonal direction have the lowest probability, and the current cell has the highest probability. If there is a food in an adjacent cell, the probability of going to that cell is increased. If there is a predator in a cell, the probability is decreased (this "fleeing" from a predator takes priority over hunger). For predators, potential food also influences the probability. For example, a

wolf will have a high probability of going to a cell with a lot of grass since it knows that rabbits will also go to that cell.

Output

There are two options for printing output. The user can either print the entire grid on each time step (used for debugging), or print the number of each animal at each time step. Printing the animal numbers is used for plotting and analyzing results.

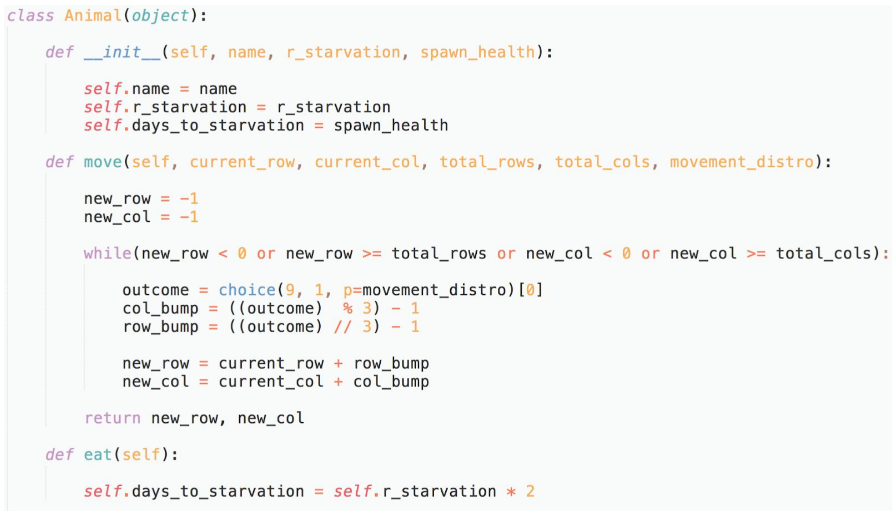
Verification

In this project, several different verification techniques were used to ensure that the software created is as intended (and matches the project conceptualized in the start).

1. Structured and modular programming

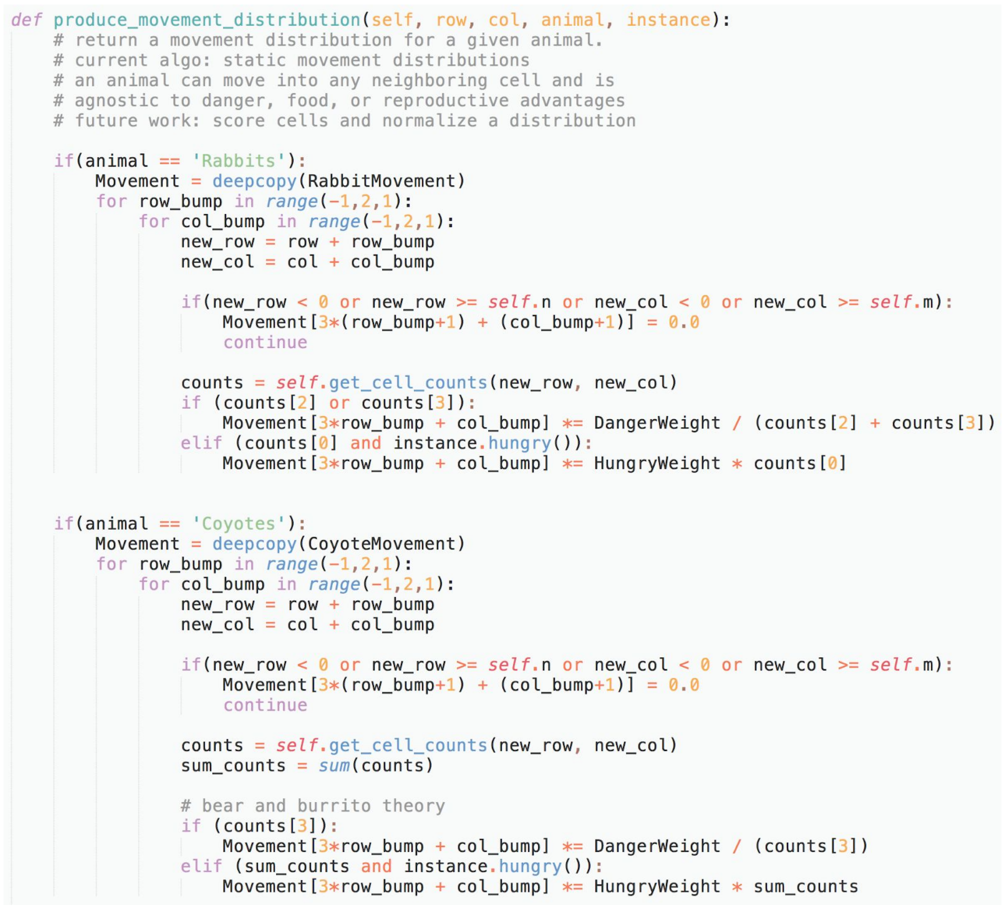
Modular and object oriented programming conventions were strictly followed so that functionality was split up sufficiently and hence, the program could be verified on a modular basis.

For example, our code currently encompasses 3 animals (Rabbits, Coyotes, Wolves). Each animal type is just an instantiation of the animal object (pictured below). Adding another animal to the simulator simply requires another instantiation of this class.



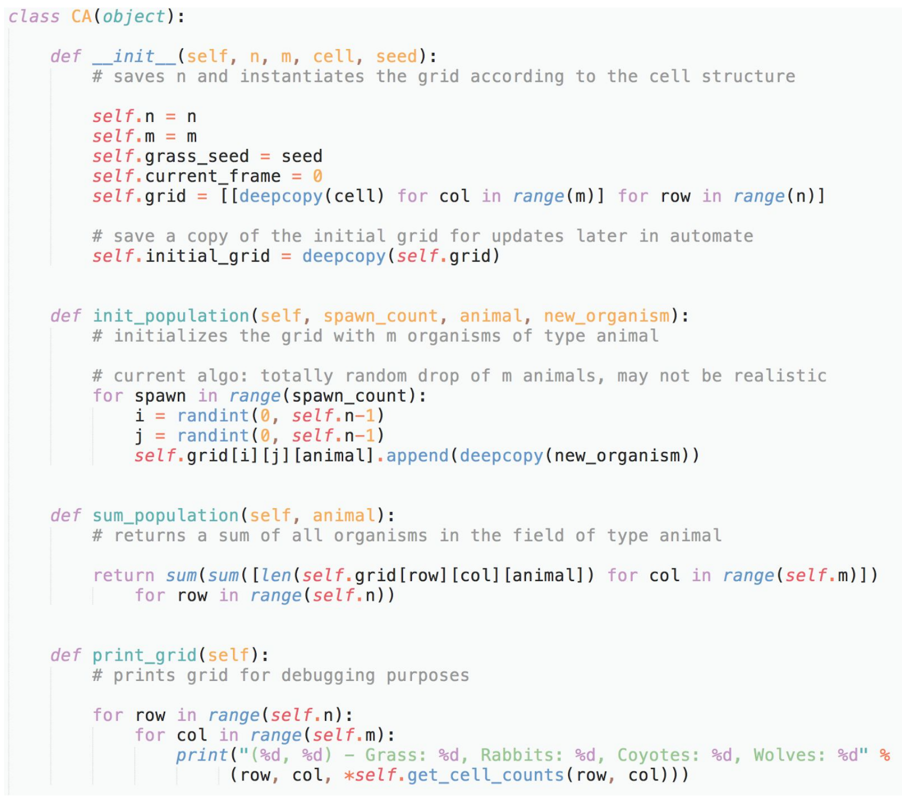
```python
class Animal(object):

    def __init__(self, name, r_starvation, spawn_health):

        self.name = name
        self.r_starvation = r_starvation
        self.days_to_starvation = spawn_health

    def move(self, current_row, current_col, total_rows, total_cols, movement_distro):

        new_row = -1
        new_col = -1

        while(new_row < 0 or new_row >= total_rows or new_col < 0 or new_col >= total_cols):

            outcome = choice(9, 1, p=movement_distro)[0]
            col_bump = ((outcome)  % 3) - 1
            row_bump = ((outcome) // 3) - 1

            new_row = current_row + row_bump
            new_col = current_col + col_bump

        return new_row, new_col

    def eat(self):

        self.days_to_starvation = self.r_starvation * 2
```

Similarly, each interaction that occurs between the different species is also a separate module. The module pictured below handles movement of species from one cell to another. Since each functionality is part of its own module, it was possible to test and verify the accuracy and performance of each module (see below).



```python
def produce_movement_distribution(self, row, col, animal, instance):
    # return a movement distribution for a given animal.
    # current algo: static movement distributions
    # an animal can move into any neighboring cell and is
    # agnostic to danger, food, or reproductive advantages
    # future work: score cells and normalize a distribution

    if(animal == 'Rabbits'):
        Movement = deepcopy(RabbitMovement)
        for row_bump in range(-1,2,1):
            for col_bump in range(-1,2,1):
                new_row = row + row_bump
                new_col = col + col_bump

                if(new_row < 0 or new_row >= self.n or new_col < 0 or new_col >= self.m):
                    Movement[3*(row_bump+1) + (col_bump+1)] = 0.0
                    continue

                counts = self.get_cell_counts(new_row, new_col)
                if (counts[2] or counts[3]):
                    Movement[3*row_bump + col_bump] *= DangerWeight / (counts[2] + counts[3])
                elif (counts[0] and instance.hungry()):
                    Movement[3*row_bump + col_bump] *= HungryWeight * counts[0]


    if(animal == 'Coyotes'):
        Movement = deepcopy(CoyoteMovement)
        for row_bump in range(-1,2,1):
            for col_bump in range(-1,2,1):
                new_row = row + row_bump
                new_col = col + col_bump

                if(new_row < 0 or new_row >= self.n or new_col < 0 or new_col >= self.m):
                    Movement[3*(row_bump+1) + (col_bump+1)] = 0.0
                    continue

                counts = self.get_cell_counts(new_row, new_col)
                sum_counts = sum(counts)

                # bear and burrito theory
                if (counts[3]):
                    Movement[3*row_bump + col_bump] *= DangerWeight / (counts[3])
                elif (sum_counts and instance.hungry()):
                    Movement[3*row_bump + col_bump] *= HungryWeight * sum_counts
```

Finally, the cellular automata simulation itself was split into separate functions and methods, based on the logical separation noted in the conceptual model above. There are separate functions and methods that handle initializing the grid cell, time stepping through the simulation, printing the result and grid etc. Printing the output and grid at each time step also allowed us to conduct a trace analysis of the simulation, to note the progression of the grid states (and animal populations), through all intermediary time steps.



```python
class CA(object):

    def __init__(self, n, m, cell, seed):
        # saves n and instantiates the grid according to the cell structure

        self.n = n
        self.m = m
        self.grass_seed = seed
        self.current_frame = 0
        self.grid = [[deepcopy(cell) for col in range(m)] for row in range(n)]

        # save a copy of the initial grid for updates later in automate
        self.initial_grid = deepcopy(self.grid)


    def init_population(self, spawn_count, animal, new_organism):
        # initializes the grid with m organisms of type animal

        # current algo: totally random drop of m animals, may not be realistic
        for spawn in range(spawn_count):
            i = randint(0, self.n-1)
            j = randint(0, self.n-1)
            self.grid[i][j][animal].append(deepcopy(new_organism))


    def sum_population(self, animal):
        # returns a sum of all organisms in the field of type animal

        return sum(sum([len(self.grid[row][col][animal]) for col in range(self.m)])
            for row in range(self.n))


    def print_grid(self):
        # prints grid for debugging purposes

        for row in range(self.n):
            for col in range(self.m):
                print("(%d, %d) - Grass: %d, Rabbits: %d, Coyotes: %d, Wolves: %d" %
                    (row, col, *self.get_cell_counts(row, col)))
```

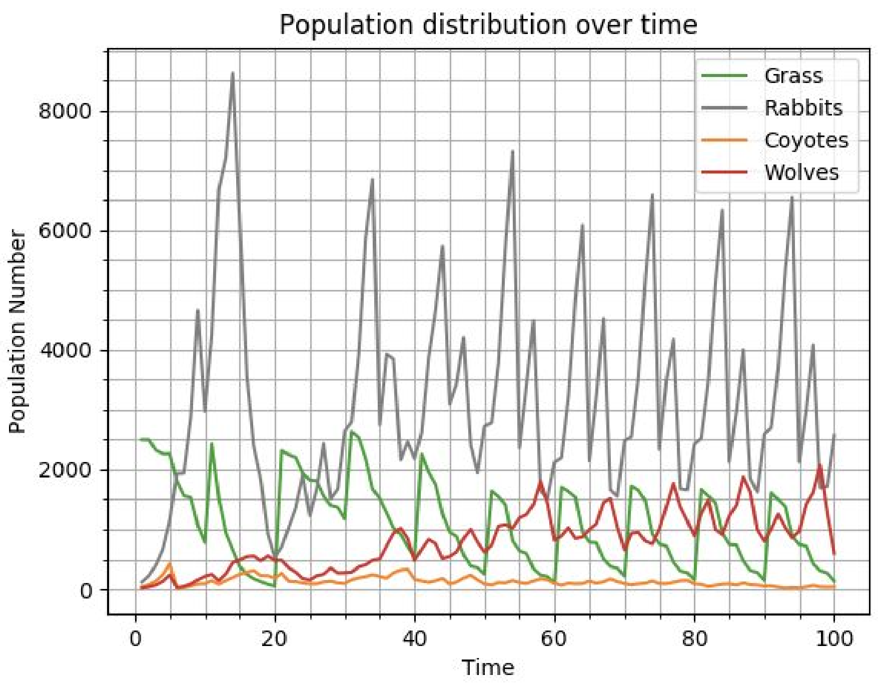
2. Validate "reasonableness" of results as variables are varied (and trace analysis)

Another verification technique used was to vary the initialized variables and ensuring that the output matches the logical inferences from the conceptual model. For example, the simulation was run for low success of wolf hunting (the wolf has a low chance of successfully hunting his prey) and a high success of wolf hunting. The traces of the population of each animal for these two cases is pictured below. In the first case, when the wolves have a low chance of successfully hunting, we see that their average numbers are kept relatively low (1500 wolves on average) and a large rabbit population is sustained. The rabbit population is controlled by the scarcity of food, as peaks and troughs of the rabbit population match that of the grass growth.

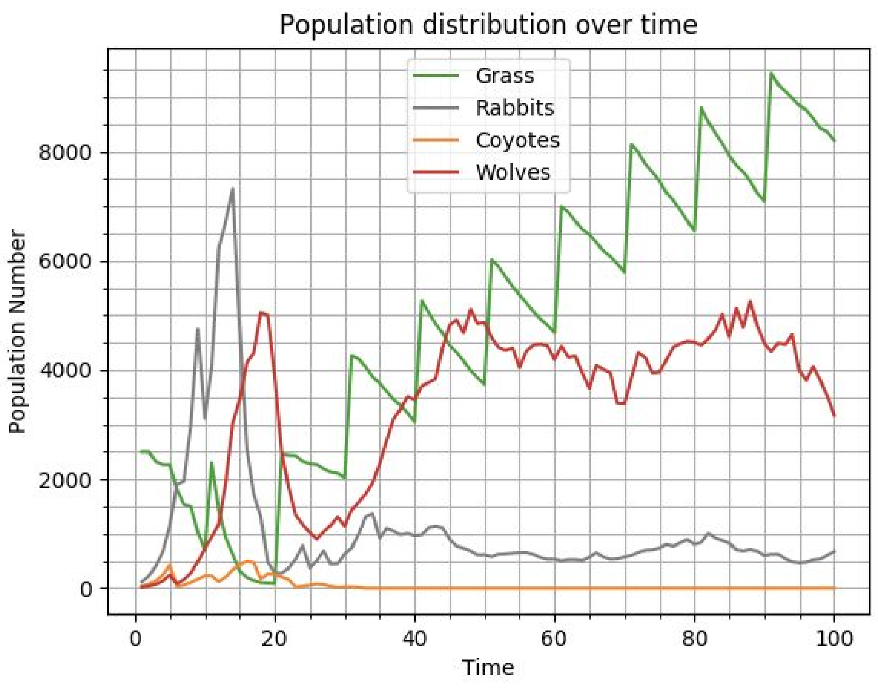
When the hunting success variable is varied, and the wolves are made more successful, the simulation behaves exactly as expected. The wolf breed is more successful and the average number of wolves now is around 4000. In fact, the wolves are so successful in hunting, that they eradicate all the coyotes and almost kill off all the rabbits. Due to the low number of rabbits, the grass growth rises steadily.

Hence, it could be verified that the hunting mechanism works as expected on the conceptual model. A similar analysis was conducted to verify the breeding and grass growth functionality.

Low Wolf Hunting Success

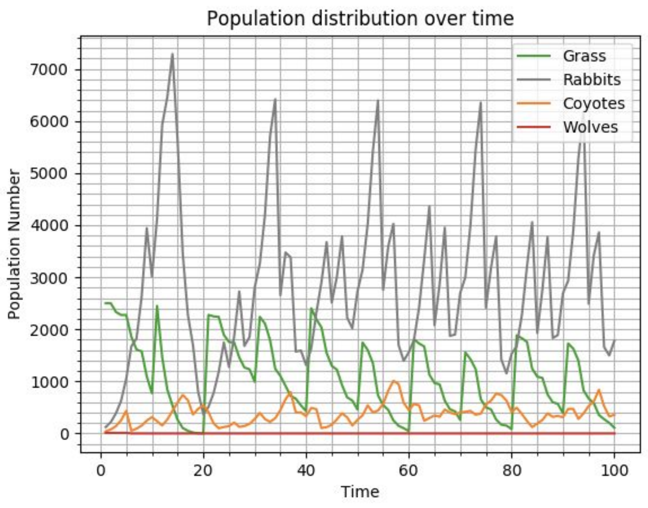


High Wolf Hunting Success



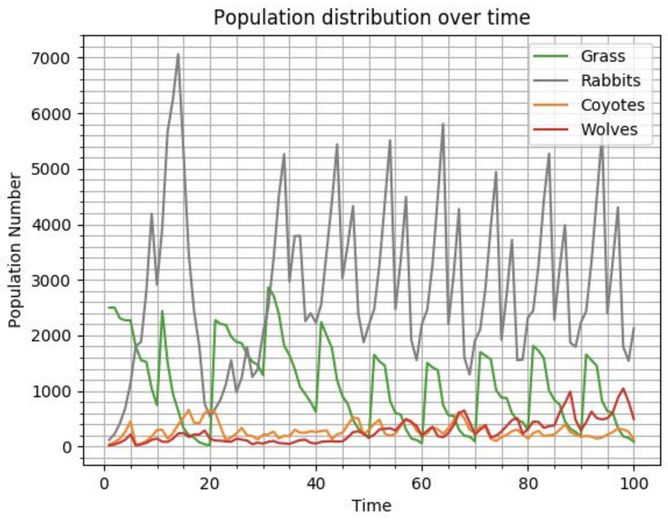
3. Test and validate simple configurations

The simulation was also tested against known outputs of some configurations. Since the only functionality that allows populations to sustain and increase is breeding, different values of breeding were simulated. The results were exactly as expectedrrq. With no breeding, the organism went extinct and its prey thrived (because there were no predators for that prey). With very high breeding, the organism thrived but its prey went extinct because now the predators



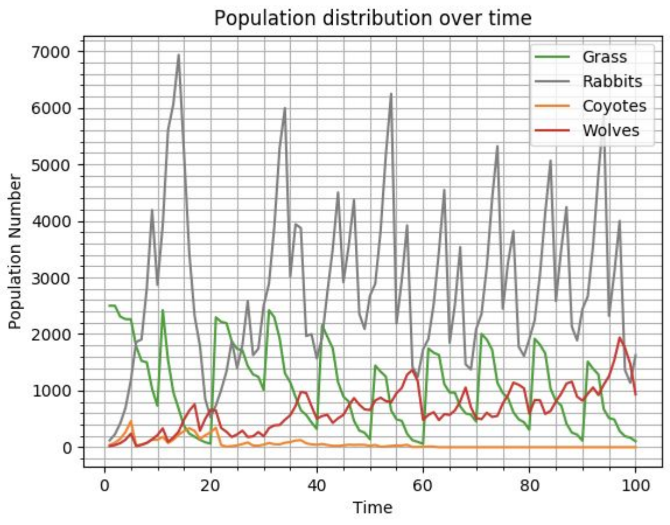
No Wolf Breeding

In this case, we can see that the wolf population starts at 10, and after a couple of time steps, immediately goes to 0. This is due to the fact that breeding is the only mechanism with which the population can increase. Because of age and starvation, the population can only decrease with no wolf breeding. Grass, Rabbit and Coyote ecosystem is still sustained however.



Low Wolf Breeding

With an increase in wolf breeding, we can see that the wolf population is sustained (unlike the previous case where they went extinct). In this case the breeding is sufficiently high enough so that all organism populations are sustained.

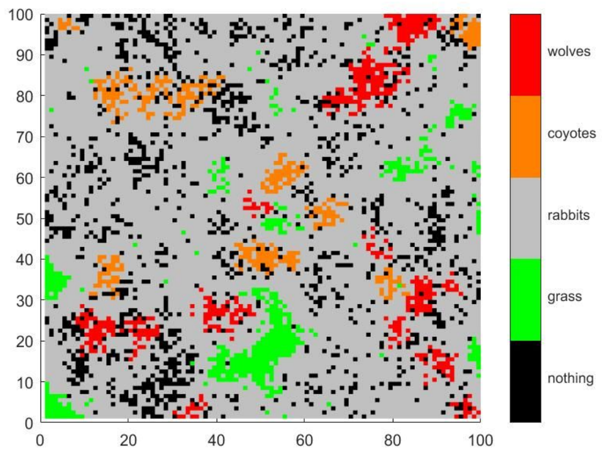


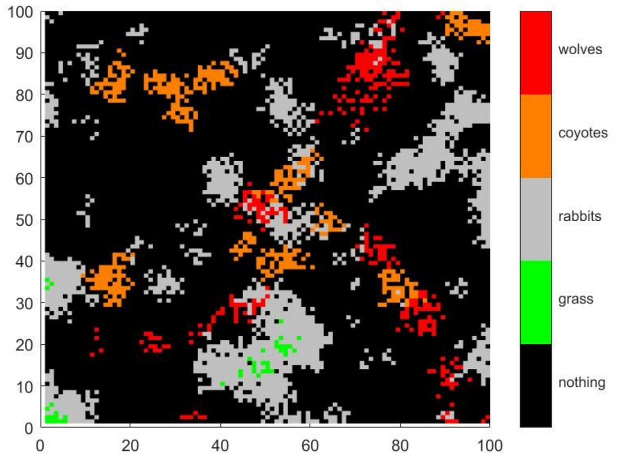
High Wolf Breeding

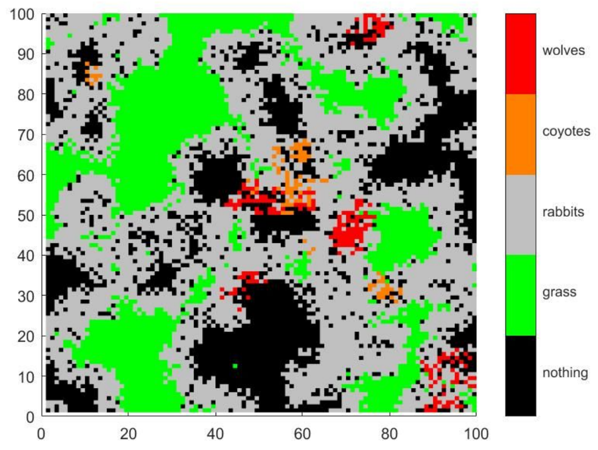
In this case, there is very high wolf breeding. The consequence of this is clear as the average wolf population is much greater than the other two cases. In fact, there are too many wolves that prey on coyotes, that the coyotes go extinct. Ence, in this case, the populations could not be sustained due to wolf breeding being too high.

4. **Model animation**

Chaotic Behavior





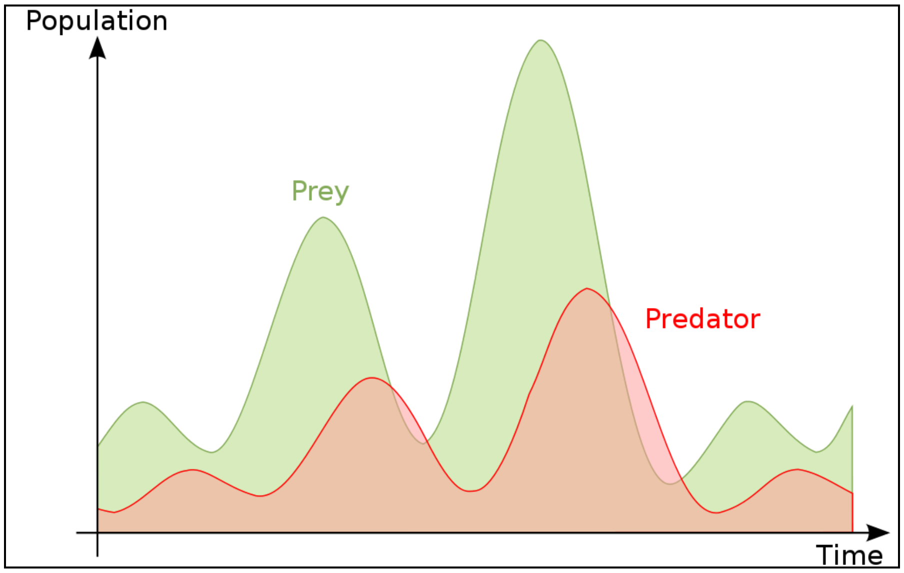


The above figures show positions of different species at time step 100 for different random number seeds. All parameters are the same except for the seed. Note that multiple species can occupy a grid location, so we prioritized predators over prey. We can see that by changing the initial conditions, the resulting positions vary greatly. This is chaotic behavior, which is what we see in general for our model. This does not mean populations cannot be sustained, just that positions of species are unpredictable.

Validation:

1. **Compare to the Lotka-Volterra Equations**

Population dynamics are commonly modelled using the Lotka-Volterra equations [4], also known as the predator-prey equations. They are simply a set of two nonlinear differential equations that describe predator-prey interactions. An example of the behavior of the Lotka-Volterra equations is shown below:

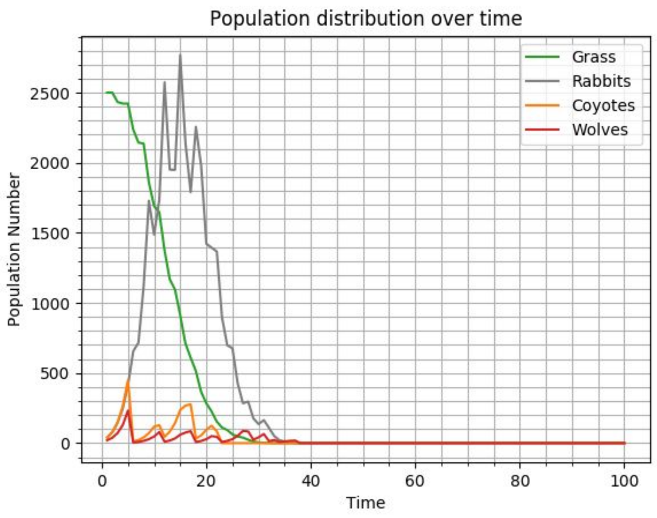


The above figure shows a common behavior; as the prey population rises, the predator population rises as well and visa versa. Note that when the predator population rises to a certain extent, the prey population will begin to fall and the predator population will subsequently drop as well. The results of our simulation program exhibit similar behavior. In fact, all trac graphs above display a version of this behavior.

2. Compare to Behavior of Food Chain

We also validated the model based on research of food chain extinction [5]. According to food chain extinction theory, if the organism on the bottom of the food chain goes extinct, then the the extinction will slowly travel upwards and all animals will go extinct in that order, because of the endless cycle of predators not having any prey. In our simulation, grass is at the bottom of the food chain, then rabbits who prey on grass, coyotes who prey on rabbits and wolves who prey on both rabbits and coyotes.

In our simulation run below, the grass growth rate is set to something very low. So it goes extinct first. Then because the rabbits have no food, they also go extinct. Now, the coyotes have no prey and they go extinct and finally the wolves go extinct. The simulation matches the food chain theory perfectly and hence, validates the predator-prey behaviour of our simulation.



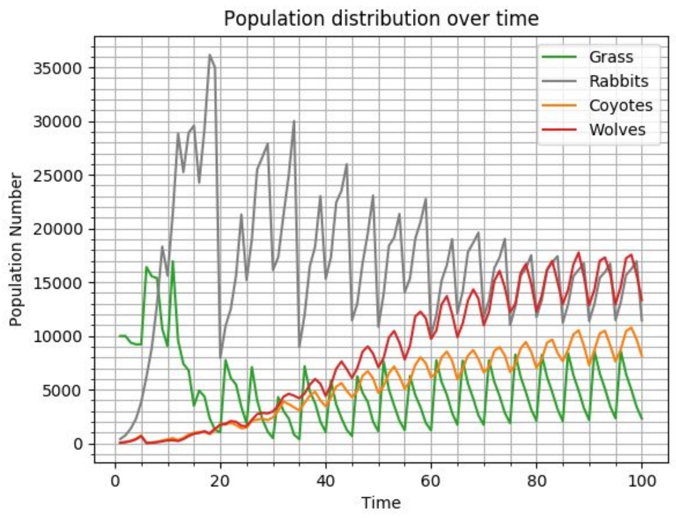
Analysis:

1. Relationship with grass growth parameter

To meet the objectives of this simulation, the grass growth rate was varied, to see how different quality seeds will affect the animal populations sustained.

The results showed that with grass that grows once in 20 days, the growth is too slow. By the end of the 20 days, the rabbit population is quite low, which makes the coyotes and the wolves go extinct. For the rest of the time steps, only a grass and rabbit population is sustained. With medium grass growth, all populations are sustained and no animals go extinct. The average populations of the animals are also reasonable. With high grass growth, the animal populations are also sustained, but the average numbers of the populations are much higher. So the highest quality grass seed can still sustain the populations but the higher numbers might not be desirable (depends on what population numbers the client expects).

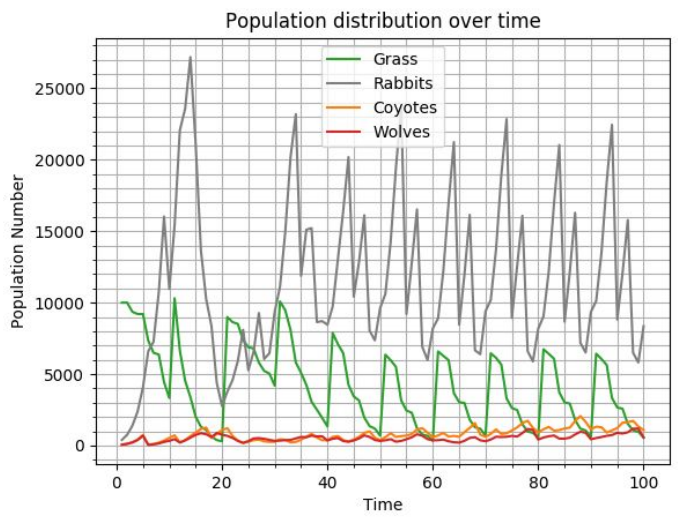
Another interesting result from this analysis is that even though the grass growth is doubled and quadrupled, the average number of grass does not seem to increase/decrease significantly. This makes logical sense, because it is a balanced, sustainable ecosystem. So if there is a large amount of grass, more rabbits are sustained, who will eat the larger amount of grass, hence controlling the magnitude.



High Grass Growth

G = [1, 5]

Grass:	5034.23
Rabbits:	16755.22
Coyotes:	5250.54
Wolves:	8264.93



Medium Grass Growth

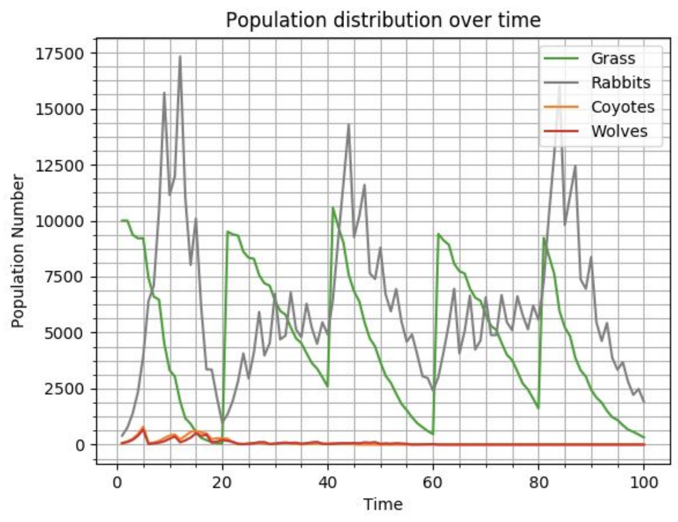
G = [1, 10]

Grass: 4200.97

Rabbits: 11469.33

Coyotes: 799.94

Wolves: 525.27



Low Grass Growth

G = [1, 20]

Grass: 4692.93

Rabbits: 6142.46

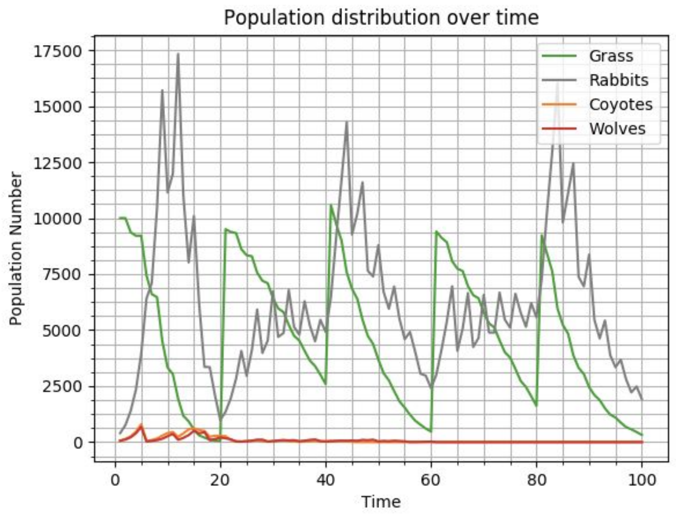
Coyotes: 81.71

Wolves: 71.44

2. Relationship with grass density parameter

The grass density (or rather, number of seeds per plot of grid) was also changed. It behaved very similarly to increasing the quality of seeds. As the density is increased, there is more grass in the ecosystem, which increases the animal populations that it can sustain. Just like the case with the grass growth, if the density is too low, the coyotes go extinct due to a low rabbit population, and then the wolves go extinct because the coyotes went extinct.

A more important result, however, is the comparison of the quality of the seeds to the density of the seeds. When the quality of the seeds were doubled, the average populations sustained were much higher than when the number of seeds were doubled. Logically, this is as expected, because with twice the seeds, there is more competition between the grass seeds for nutrients to grow. This is a very important result for our client, because it shows that investing in a seed that's got twice the growth capability is better than planting twice the number of grass seeds. This is an important factor to consider in the cost optimization that they plan to do.



Low Grass Density

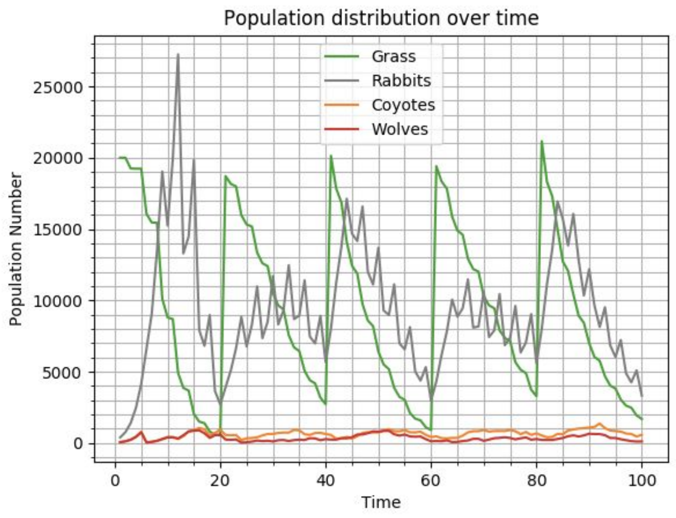
G = [1, 20]

Grass: 4692.93

Rabbits: 6142.46

Coyotes: 81.71

Wolves: 71.44



High Grass Density

G = [2, 20]

Grass: 9355.86

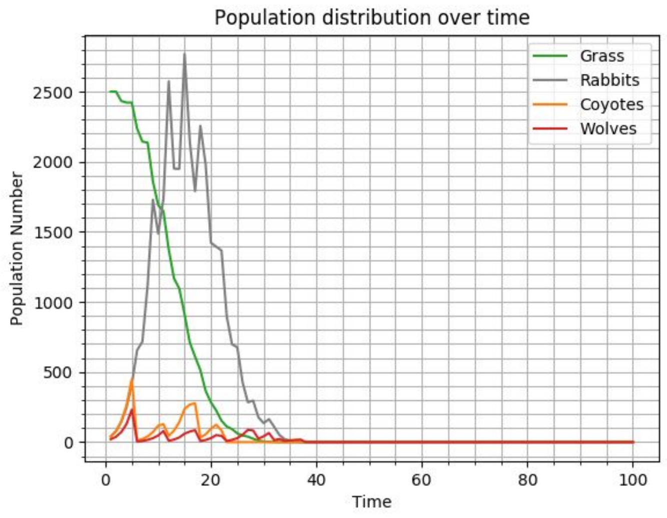
Rabbits: 9233.45

Coyotes: 644.07

Wolves: 350.7

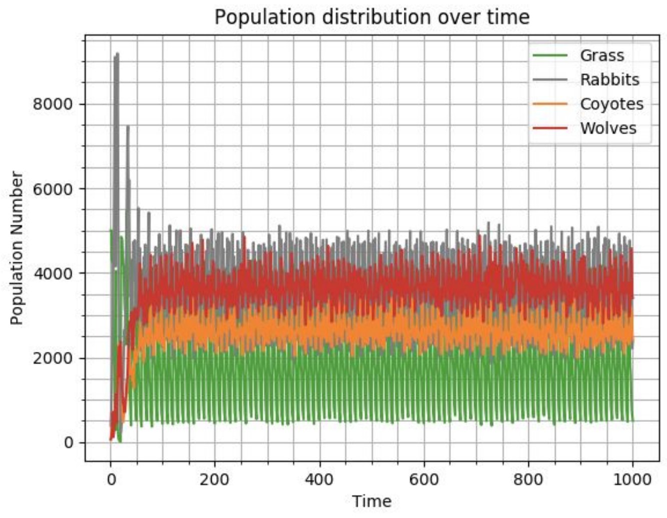
3. Cellular Automata stability and complexity analysis

Because this is a cellular automata with complex, stochastic rules, it is expected to see complex behavior. With changes in initial condition, we can note the different behaviors that occur (and the logical reasoning for these behaviors). A fixed point is achieved when all animals go extinct. Periodic behavior is more common, as most initial conditions will cause the ecosystem to stabilize at different population numbers for the animals. Chaotic behavior is observed when initial conditions given are extreme, which leads to random behaviors.



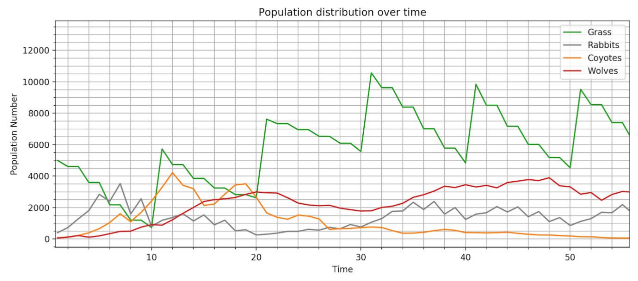
Fixed Point Behavior

When grass doesn't grow (initial condition of grass growth is 0), the system will always move towards an extinction of all species. As explained above, without any population of the lowest level of the food chain, to higher levels can be sustained.



Periodic Behavior

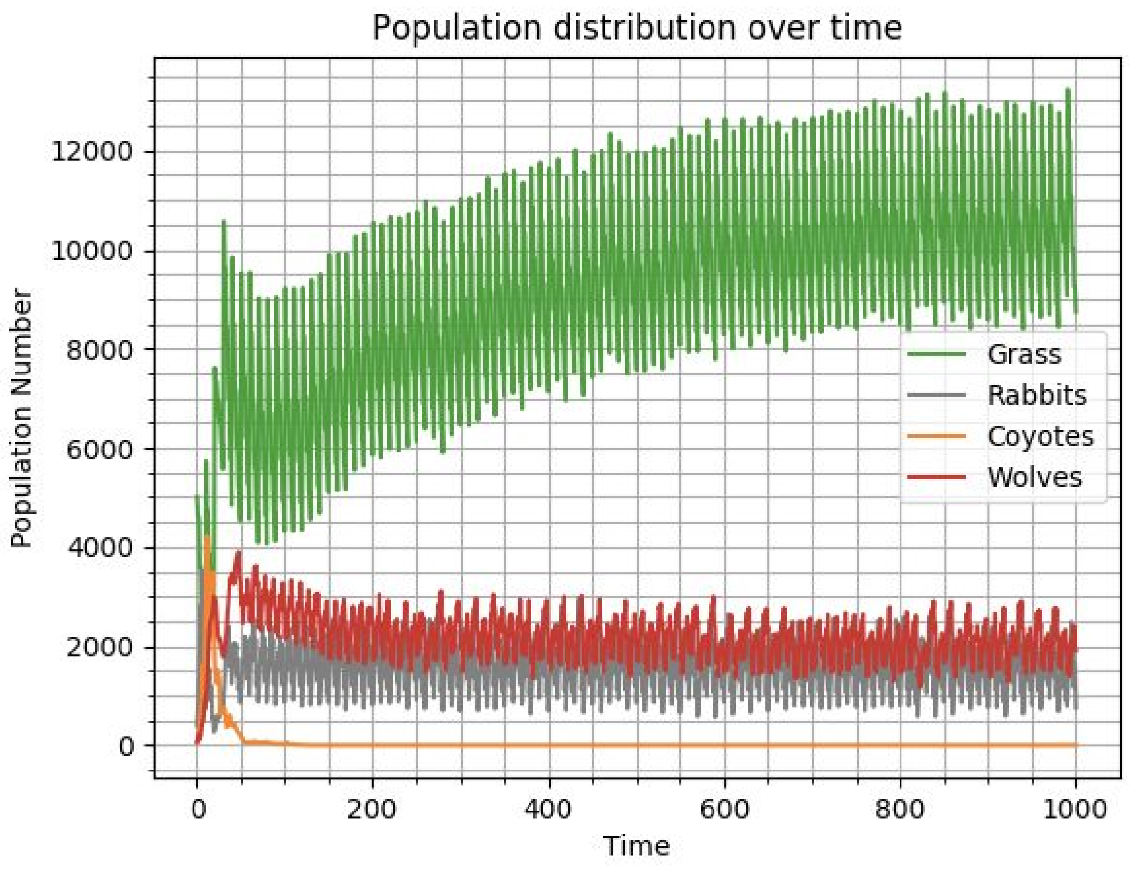
Periodic behavior is obtained when all the values are “normal” (breeding, hunting and grass growth are all moderate values). The population numbers for each organism oscillates between values, but the mean is largely the same.



Chaotic Behavior

Chaotic behavior is observed when the initial conditions are changed, so that any one feature is very high (or very low). In this simulation, the breeding rates and hunting rates were set to very high values, which leads to chaotic (and almost random) results.

In the simulation below, we can see that the simulation is complex in regards to the population numbers of the animals. This single simulation starts off being chaotic and eventually stabilizes to a periodic change in population numbers. The coyotes however, go extinct, and achieve fixed-point stability with regards to population numbers.



Conclusion

This report describes the simulation of animal behaviors when certain grass species' are placed introduced into the environment. The simulation considers only wolves, coyotes and rabbits to model predator and prey behaviors. The wolves hunt coyotes and rabbits, the coyotes hunt rabbits, and the rabbits consume grass. A conservation agency is the target client for our software since the need to plant certain types of grass may be a necessity in order to sustain animal populations. We used the stochastic cellular automata method to model this physical system. We described how we implemented the code, which was done in an object oriented style so that future development of the code would be easy and so that the code would be easy to test. We validated our system by comparing our results to theory derived by Lotka and Voltara, and by comparing our output to actual data collected from observing food chains. We showed that populations can be sustained (i.e., populations of different animals do not grow unbounded and do not go extinct) using certain grass seed types, and we showed that our system can exhibit all types of behavior (e.g., chaos).

References

1. http://www.angelfire.com/ca5/magic1/Pred.html
2. https://www.ncbi.nlm.nih.gov/pubmed/17922704
3. https://extension2.missouri.edu/g9412
4. Lotka, A. J. (1910). "Contribution to the Theory of Periodic Reaction". *J. Phys. Chem.* **14** (3): 271–274.
5. https://pdfs.semanticscholar.org/ba0d/bfe988653096be74c4935e338b7c6548ccdb.pdf