

Anish & Ishaan

Science Fair Lab Report

June 14, 2016



Overview

The purpose of this experiment is to determine whether or not, and to what extent of accuracy we can efficiently simulate real ecosystems. In this report we will discuss the creation of the program, some biological factors that were taken into consideration, why this software can prove to be helpful in the future, and where we see this software to be headed.

What is Computational Biology?

Computational Biology at its core is using computation in the field of biology. An example of this is in DNA strands; one can use a computer program in order to find a specified DNA strand, which would be very troublesome for a normal human being. In general, computational biology is used for simulating, searching, predicting, etc. According to Wikipedia, computational biology is “the science of using **biological** data to develop algorithms and relations among various **biological** systems”.

Program Commands:

- # → Number of generations you would like to simulate at the moment. Only the nth simulation will actually be printed, for efficiency purposes.
- -1 → open up menu
 - M → Show the peaks of each population over the amount of generations passed so far
 - A → Show average population for each animal over all the generations passed so far
 - G → Open menu for outputting all data over every generation
 - 1 → Print all data values in array format, for further use in another programming language (ex. Python for data mining)
 - 2 → Print all data values in table format for copying and pasting to excel, or for easier viewing
 - ** Data is printed out with one generation per line, and each line shows that generation's population in the following order
 - Grass
 - Grasshoppers
 - Rabbits
 - Mice
 - Snakes
 - Kookaburras
 - Eagles
 - After one of the preceding commands ('M', 'A', 'G') are selected, an option will come up to enter a #
 - If # != 0, continue inquiring (choose another letter)
 - If # == 0, go back to option for entering number of generations you want to simulate

Introduction:

The idea of simulation of ecosystems was initially inspired by Conway's Game of Life. Conway's Game of Life is a cellular automaton created by British mathematician John Conway. His simulation determined whether or not a cell would survive a generation, the movement of cells, and the birth of cells over generations, based on some set of parameters. In the end, his program outputted different, elegant patterns depending on how many, and where initial cells were configured. While reading an article on this, we were inspired to use this ideology in the field of computational biology; by simulating entire ecosystems, rather than just cells!

Hypothesis:

We predict that one can efficiently and accurately model/simulate real ecosystems using computation and the manipulation of biotic/abiotic factors that affect the ecosystem.

Project Process:

The task for creating a computer system to represent an ecosystem took an immense amount of time, and the process can be viewed as four different phases; research, programming, debugging, and data analysis

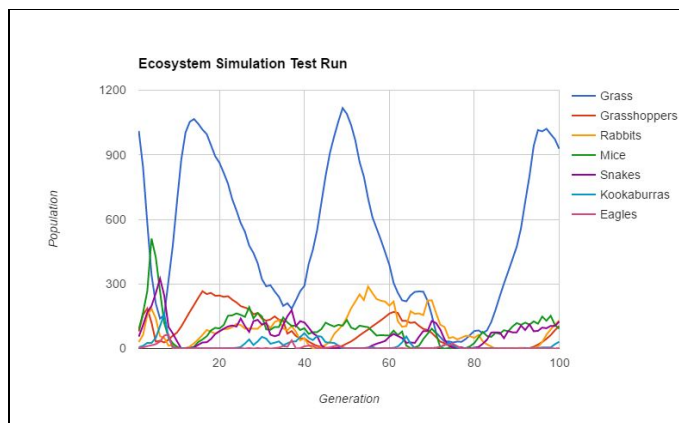
Research: The first phase - research - was possibly the most important part of the entire experiment. In order to build a program to emulate an ecosystem, we needed to truly understand the nature and behaviours behind them. Some specific research we did is noted in the research conducted section.

Programming: This phase was the most time consuming. Using computer science and programming skills, we had to first implement the barebones of the program (ex. Set up the environment, add in the different animals). The next task was to add in all the parameters and

processing functions. Lastly we worked on the I/O, keeping in mind that we needed to create an easy and accessible way to access data and statistics.

Debugging: Immediately following the programming phase, we had to do a lot of debugging, essentially the “experimentation/testing” part of the project. At first we noticed that the initial parameters were not as accurate as we’d liked them to be, so optimizing them became our goal. We manipulated variable like amount food, number of offspring, initial generation amounts, etc. until we felt we could get as close as possible. Along the way we also discovered several “bugs”, such as the eagle giving birth to kookaburras, which resulted in the eagle dying in the same generation (no mates).

Example data from debugging phase:



Although the grass seems to survive fine, the other species were a mess

Data Analysis: The final part of the process was data analysis. During this phase we observed multiple sets of data, but ended up focusing on just one (for simplicity’s sake). Visit data analysis section for more.

Challenges:

Clearly, simulating an entire ecosystem **accurately** is not a simple task. Challenges were consistently faced in computer science and biology throughout this project, both of which had to be tackled in an efficient and intelligent manner.

Biological Challenges:

The greatest challenge faced in a biological aspect without a doubt would be configuring the most accurate set parameters, to best represent a real ecosystem. In order to do this we had to research

and experiment. Based on our scientific knowledge of ecosystems, we sorted out some initial numbers based on trophic level, ecological niche, observations, patterns in nature, and the number of limiting and encouraging factors present. Then, through hundreds of generations of experimentation, we noticed some errors in the representation or issues with the generation, and slowly tweaked the statistics, optimizing the accuracy piece by piece.

Additionally, prior to experimentation we realized we would have to analyze data from other sources, of actual ecosystems. One significant experiment that we observed was the fox and rabbit experiment. Here we noticed different trends, cycling of populations, dependency and interconnection between species, etc. This proved to be necessary during our data collection/optimization phase. Again, we collected many samples of data over hundreds of generations, graphed this data, and based our accuracy fairly relative to the graph/trends of the fox/rabbit experiment.

The last hurdle we faced was determining which factors and parameters should we be able to optimize ourselves, in order to obtain the closest accuracy. Once again, through experimentation and research we determined the following variables that made the most sense to be changeable: ecosystem size, animal food/birth range, number of offspring, amount of food consumable per found, number of mates, amount of food required, initial population range, and frequency of catastrophes.

Computational Challenges:

Aside from having a solid understanding of interaction and features of an ecosystem, simulation also requires a fair amount of computer science and mathematics. Firstly, being a natural environment, controlled probability and modular arithmetics were used to compute initial generations (ex. How many patches of grass, and how large each individual patch of grass will be, to start off). Second, basic grid traversal was required. As the entire simulation is represented with numbers in a grid, we had to be able to manipulate these grids to produce results (grids being 2 dimensional arrays). Third, we needed to be able to handle I/O. While this is mostly a programming challenge rather than computation, being a software intended to be used by others, implementation for simple interaction between the program and user was a must. Fourth, basic mathematics were used to calculate the simulations. Some examples of calculations were population counting, population averages, ranges for traversals, maximum and minimum values, and boolean logic. Lastly but not least, to power the entire program, basic data structures knowledge proved to be helpful. In general, single and multidimensional arrays and vectors were used.

Biological/Ecological Research Conducted:

As mentioned in the “challenges” portion of this report, in order to reach the greatest degree of accuracy as possible, we would need to have a solid understanding of biology, and how ecosystems work. The following are some terms, ideology, and processes we had to research prior to building the software.

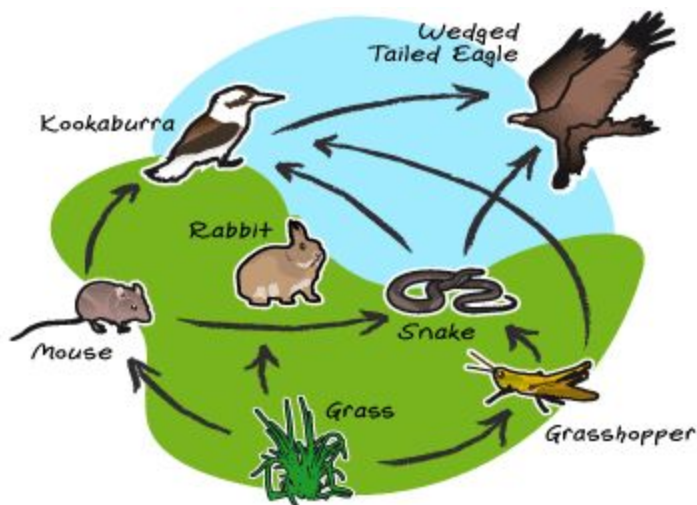
Trophic Levels: The level at which a species is located in a food web. Generally speaking, the further out the trophic level for a species is the more it’s population depends on others, and the further in a species is, the more influential ecological niches it has. Heterotrophs consume other species while autotrophs produce their own food/energy.

Ecological Niches: The job a species plays in it’s ecosystem. The greater the ecological niche, the more important it is to the environment, thus the more it is depended on

Population Cycling: where populations rise and fall over a predictable period of time.

Proportionality of Ecosystem Size to Stability:

Ecosystem Simulated:



Simulated Data Sample:

The following data was collected over a sample of 100 generations!

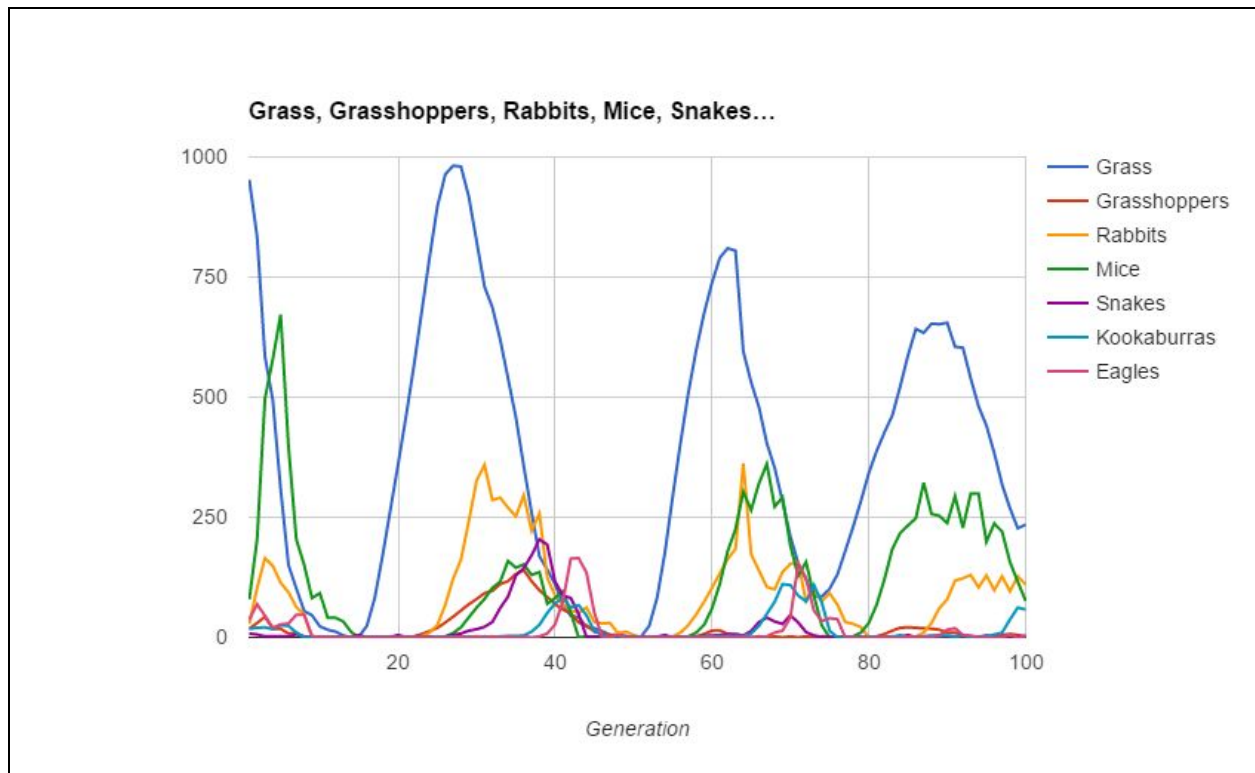
Generation	Grass	Grasshoppers	Rabbits	Mice	Snakes	Kookaburras	Eagles
1	953	18	28	79	8	18	37
2	834	30	103	204	6	20	69
3	585	43	165	497	2	21	45
4	494	18	149	579	2	17	20
5	309	18	114	672	2	25	27
6	151	8	94	398	2	24	29
7	102	10	63	204	6	10	47
8	55	0	48	151	2	0	47
9	46	2	0	82	0	0	3
10	23	0	2	92	1	0	0
11	16	2	0	41	0	0	0
12	12	0	2	41	1	0	0
13	4	2	0	33	0	0	0
14	0	0	2	11	1	0	0
15	2	0	0	0	5	0	0
16	24	2	0	0	0	0	0
17	82	0	2	0	1	0	0
18	170	2	0	0	0	0	0
19	266	0	2	0	1	0	0

20	362	2	0	0	5	0	0
21	461	0	2	0	0	0	0
22	566	2	0	0	1	0	0
23	679	7	2	0	0	0	0
24	792	13	10	2	1	0	0
25	898	20	26	0	0	2	0
26	964	31	68	2	1	0	0
27	982	43	122	9	5	2	0
28	980	56	162	21	8	0	2
29	917	69	242	42	14	2	0
30	824	80	327	62	17	0	2
31	730	92	359	80	22	2	0
32	687	97	286	103	32	0	2
33	620	111	291	116	61	2	0
34	539	117	270	159	85	3	2
35	458	132	252	145	129	3	0
36	357	140	296	152	144	4	2
37	260	117	222	130	171	12	0
38	169	98	258	136	204	26	2
39	141	84	124	70	193	51	8
40	111	69	87	82	116	71	27
41	68	58	69	96	87	82	76
42	45	50	56	51	82	62	164

43	42	31	51	0	49	67	165
44	22	25	62	2	1	46	135
45	19	12	32	0	2	13	52
46	13	8	29	2	0	5	0
47	5	5	30	0	2	1	1
48	6	3	9	2	0	3	5
49	3	1	12	0	2	0	0
50	0	3	5	2	0	0	0
51	2	0	0	0	1	0	0
52	24	2	0	0	0	0	0
53	82	0	2	0	1	0	0
54	176	2	0	0	5	0	0
55	294	0	2	0	0	0	0
56	406	2	10	0	1	0	0
57	510	0	26	2	0	0	0
58	599	2	48	10	1	0	0
59	675	7	74	26	3	2	0
60	738	14	103	60	4	3	2
61	790	15	133	111	5	3	0
62	810	7	163	179	7	3	2
63	805	4	184	225	7	3	0
64	595	3	362	303	4	3	2
65	531	0	173	265	13	8	0

66	479	2	139	321	32	24	0
67	404	0	105	361	41	49	2
68	353	2	100	272	33	74	10
69	283	0	136	292	28	111	14
70	210	2	153	191	46	109	41
71	154	0	156	128	31	87	153
72	121	2	75	157	11	75	123
73	86	0	98	87	4	109	56
74	85	2	75	25	1	73	34
75	101	2	92	0	1	14	40
76	131	0	68	2	0	0	38
77	180	2	32	0	1	0	0
78	230	0	28	2	0	0	0
79	284	2	21	10	1	0	0
80	342	0	0	30	0	2	0
81	388	2	0	68	1	0	0
82	427	7	2	121	0	0	0
83	462	14	0	184	2	1	0
84	522	20	2	217	1	5	0
85	587	21	0	233	5	0	2
86	642	20	2	248	0	0	0
87	634	19	10	322	2	1	0
88	653	18	30	257	2	4	1

89	652	16	61	253	4	3	4
90	655	10	79	238	2	4	16
91	605	10	118	294	2	1	19
92	603	5	123	228	2	4	5
93	539	0	130	299	0	0	4
94	481	2	104	299	1	0	0
95	441	0	129	199	4	2	0
96	384	2	98	237	0	6	0
97	318	5	127	220	2	10	1
98	269	7	96	157	0	37	5
99	227	5	127	116	2	62	0
100	235	1	110	76	3	58	2



Example Data Analysis:

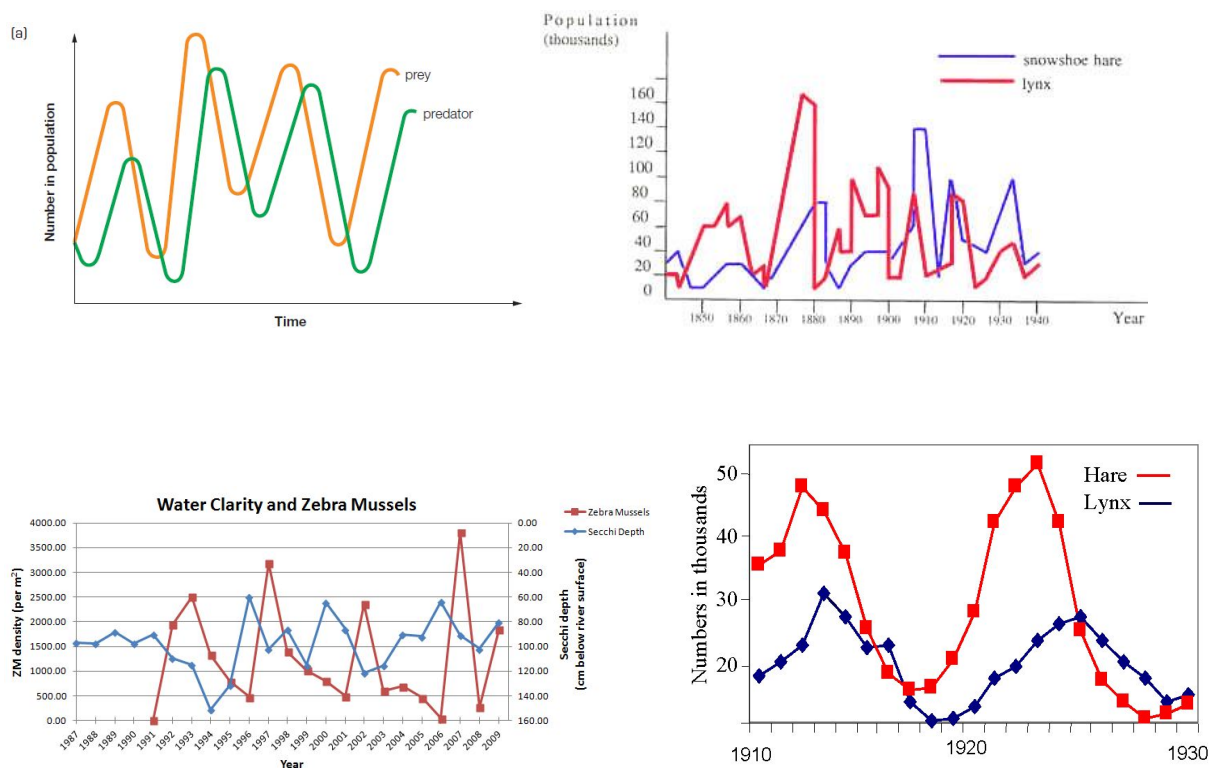
- The data accurately represents the relation between ecological niche and importance in ecosystem. For example the grass is responsible for the entire ecosystem since it is the only autotroph, and we can see the other populations follow the trends of the grass (therefore are extremely dependent on it)
- According to ecologists, there should be a proportionality between ecosystem size and stability. The ecosystem used was fairly small (7 species total), so correspondingly we see great fluctuations in the populations
- We can observe cycles in the data, therefore there are predator/prey relationships; as predator increases, prey decreases and vice versa

Animal	Average Population	Plateau at Peak (approx)	Carrying Capacity (Max)
Grass	375	750	980
Grasshoppers	19	80	640

Rabbits	84	175	375
Mice	118	260	574
Snakes	17	30	204
Kookaburras	15	50	109
Eagles	15	15	165

Examples of Real Population Graphs:

For comparison purposes.



Features of Simulation:

Automatic Data Collection

As mentioned in the program commands section, much of the data generated by the program can be retrieved. One can request for peaks in population, averages, and all the previous generation statistics (in the form of arrays or charts).

Examples of maximum and average statistics menus:

```
Enter number of generations to skip (1..n) or 0 to end: -1
Would you like to view max (M), averages (A), or generations (G)?:
M
max grass: 1434
max grasshoppers: 229
max rabbits: 303
max mice: 566
max snakes: 143
max kookaburras: 159
max eagles:180
If you would like to continue inquiring enter any integer other than 0. Else enter 0:
```

```
Would you like to view max (M), averages (A), or generations (G)?:
A
avg grass: 616
avg grasshoppers: 39
avg rabbits: 75
avg mice: 156
avg snakes: 22
avg kookaburras: 18
avg eagles:16
If you would like to continue inquiring enter any integer other than 0. Else enter 0:
```

Scalable Parameters

In order to make the program much more user friendly, the parameters for each animal's variables can easily be changed. This is because objects and general variables were wisely used. So, in the event that one notices an irregularity in the simulation, they can spend more time optimizing rather than having to deal with changing large portions of code.

Multiple Assessment Factors

```

} else if (grid[y][x] == 7) {
    eagles++;
    //eagles are capable of eating more than 1 animal per gen, so we will set it to eat a max of 2. But, they still
    must eat at least 1 to stay alive
    int eagleDigested = 0;
    for (int i = y - eagle.radius; i < y + eagle.radius && i < width; i++) {
        for (int z = x - eagle.radius; z < x + eagle.radius && z < width; z++) {
            if (i < 0) {
                i = 0;
            }
            if (z < 0) {
                z = 0;
            }

            if ( (grid[i][z] == 6 || grid[i][z] == 5 /*|| grid[i][z] == 3*/) && eagleDigested < 1) {
                grid[i][z] = 0;
                eagleDigested++;
                if (eagleDigested >= 1) {
                    food = true;
                }
            }
            else if (grid[i][z] == 7) {
                mate = true;
            }
            if (mate && food) {
                eagleLocations.push_back(temp);
                i = 100000;
                z = 100000;
            } else if ((i == y + eagle.radius - 1 && z == x + eagle.radius - 1) || (i == width - 1 || z == width - 1)) {
                grid[y][x] = 0;
            }
        }
    }
    //eagleLocations.push_back(temp);
}
}

```

Snip of code taken from program

Assessment factors in order from top-down:

- The eagles radius; the range in can hunt and mate in
- Prey the eagle can eat, in this case 6 represents kookaburras and 5 represents snakes
- Booleans for food and mate, to check if offspring will be born for the given eagle
- Check if mate and food are true, if so push to next phase
- Otherwise kill animal

The importance of have all these factors is that allows for more precision in representation of an actual ecosystem. Also, it allows for more possible alterations and combinations, which is useful for debugging and optimization

Mathematical Randomness

As mentioned before, the program is pseudo-random. This means that the program is capable of generating random initial values so that it displays a different ecosystem each time (although it is possible to change it to set values, if need be), but at the same time the randomness is controlled. With controlled randomness we can align statistics to what we find in nature, based on ranges and

domains. When combined we get pseudo-randomness, which gives us the best representation from both factors.

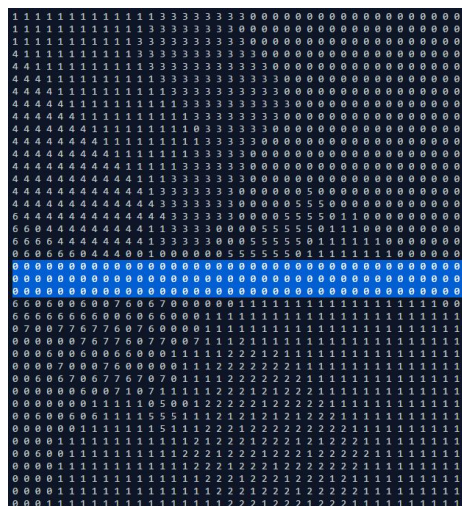
Regeneration from Extinction/Extirpation

Since we know that the smaller an ecosystem is, the greater it fluctuates, we know that for small ecosystems there is a high chance for a species to go extinct/extirpate. In fact, during the testing phase for our program, we noticed that it was quite common for a certain species to go extinct within the first 5-10 generations. Because species are highly dependent on each other in small ecosystems, with one species going extinct it led to chain reaction for other species to die out as well. In order to counteract this phenomenon, we implemented regeneration based on the rule that if a species is extinct, we can import 2 animals from that species in a random location in the ecosystem. This idea may not be too far off from a real ecosystem as when a species is close to extinction there is a chance for more/less competition, which leads to immigration or emigration of species. Overall, with the implementation of this regeneration function, we can observe ecosystems over significantly larger spans of time.

Natural Disasters

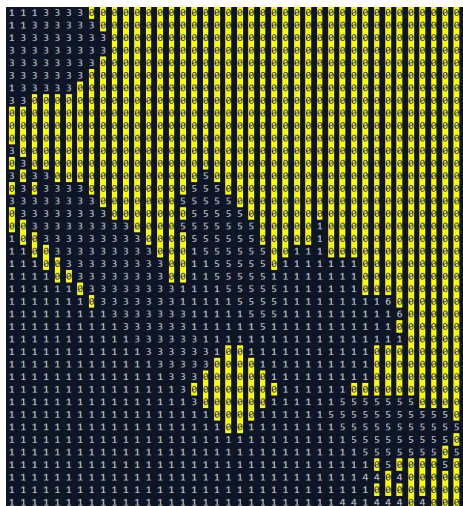
Another environmental factor that affects an ecosystem and its population are natural disasters. Although they are rare, they do happen. This is why we programmed our simulation to endure a natural disaster at least once every 100 generations. The different types of natural disasters present at the moment are fissures, and lightning strikes.

Fissures:



As we can see from the image above, a fissure occurred in the 12th row down of the ecosystem, shown as 3 consecutive horizontal lines cleared out.

Lightning Strikes:



Indicated above in yellow, there is a circular patch struck by lightning, killing everything within the radius!

How Will Our Project Benefit The World?

Time and time again, science has showed us that changes in ecosystems can have a major impact to humans and other species alike. It also has taught us that ecosystems can be fairly sensitive, as many species do not have an significant range of tolerance, are not very adaptable, and tend to depend on several other species. An example of this are situations of invasive species, such as the Asian Longhorned Beetle incident. The species of beetle was originally native to countries in Asia like Japan, but were introduced to New York around 1996, and have spread to areas in North America ever since. From then, the beetle continued to wreak havoc on many ecosystems as they lay larvae in trees (which officials tend to have to burn down to get rid of), and had no natural predators in North America. So far it is estimated that the Asian Longhorn Beetle has caused a loss of tens to hundreds of billions of dollars, as well as killing many trees which would lead to significant changes in corresponding ecosystems.

This is where we believe our project can make a difference. Rather than being able to visualize the effects and the interaction the beetle has with its environment only *after* the damage has been done, imagine if we could understand before hand. Using a simulation program similar to ours, one

could possibly generate a certain ecosystem with the addition of this species (and it's estimated parameters). All in all, we can predict what will happen to other species, the severity, patterns or trends after the introduction of the species. With this data, we can realize whether or not there is a potential threat, and possibly what actions we can take to help the situation.

Furthermore, data in general is always beneficial. Data is known for helping with decision making, understanding underlying patterns/trends, reducing or optimizing results, and learning from the relationships of different variables/factors. What's more is that running our program costs a whopping \$0, so there really is no loss!

Program Pseudocode:

Get required libraries;

Initiate data structures to be used;

Set up grid and dimensions;

Initialize some global variables;

```
natural Disaster function {  
    Choose random natural disaster;  
  
    Apply selected natural disaster to ecosystem;  
}
```

```
Class for animals {  
    Animal radius;  
}
```

Create instances/objects for each animal, using the class 'animals';

```
Print function {  
    Prints out chart  
}
```

Generation functions (one for each animal/species):

```
genAnimal {
    Choose random amount of animals to be generated, based on parameters

    Place that animal “n” times (random amount) in randomly generated positions on
the grid;
}

liveOrDie function {
    For each animal:
        Based on set parameters (radius, Findmate, Findfood), determine whether animal
reproduces for next gen;

        If animal reproduces, push its babies to designated data structure:

        Kill current animal;
}

currentGenPopulationCounter function {
    Iterate over grid:

        Count population of each animal for the current generation (will be used for
data/stats);
}

giveBirth functions {
    For each animal:

        Iterate through animal’s designated data structure:

            Give birth to a number of offspring (based on set parameters);

            Location of birth and distance from parent may or may not be random,
again based on parameters that best represented the real animal;
}

generation function {
    Reset any necessary data structures and variables;

    Iterate through ecosystem:

        Find out which animal occupies location, if any;
```

```
        If so, run liveOrDie function;
    After generating offspring and killing any to-be-dead species:

        Check if any species are extinct:

            If so, spawn two animals of the species in a random location;

    Run giveBirth function for each animal;

    Reset any necessary variables;

    Run currentGenPopulationCounter function;

    For each animal, output population in the current generation;

    Check if any natural disasters will be caused at this point:

        If so, run naturalDisaster function;
}

printChart function {
    Print all statistics, if following option is chosen: -1 → G → 1;
}

printChartFile function {
    Print all statistics, if following option is chosen: -1 → G → 2;
}

Main function {
    Set radius properties for each animal;

    Set up input;

    Take input:

        Follow the corresponding output;

    Run generation function;
}
```