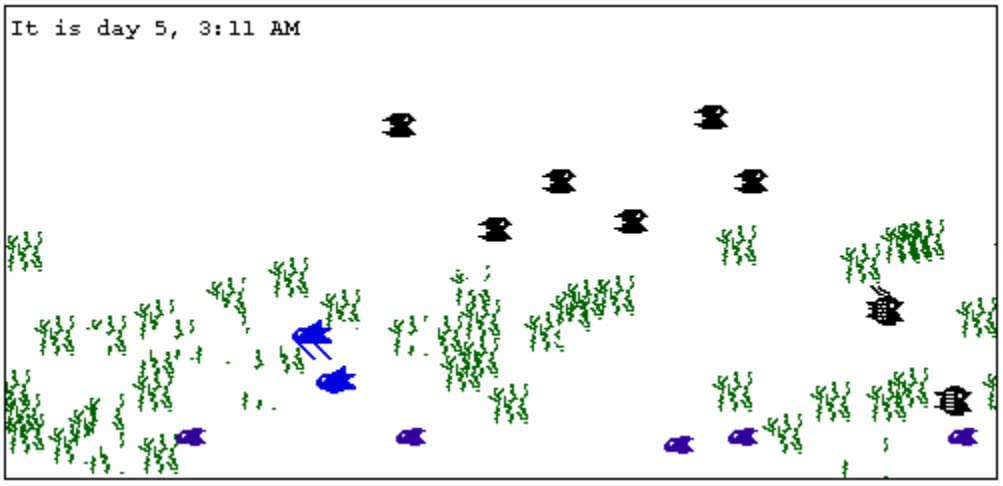
www.logofoundation.org

A LogoWriter Ecology Simulation

by

Thomas F. Trocco



Computer & Science Department Chair
Computer Teacher, Grades 2 - 8
St. Hilda's & St. Hugh's School
New York, NY

Presented at the April 13, 1993 New York Logo Users Group meeting.

Acknowledgements

Thanks to Michael Tempel, Steve Neiman, Andrew Newcomb, Eliot Widaen, and Diane Linder for their interest, suggestions, editing, and discussion. Special thanks to Michael and Diane for their suggestions on mating proximity; to Eliot for the birth placement concept; to Andy for his editing suggestions; and to Steve for his help with Apple IIGS compatibility. Extra-special thanks to Tyler Barnes, Tracy Blount, Vanessa Cook, Matthew Graves, Maritza Schaeffer, Austin Shau, and Laura Stickney for their presentation at the Logo Users Group.

Introduction

With the release of an ever-increasing number of simulation programs (e.g., *Sim Ant*, *Sim Earth*, etc.), students have much more to work with than just the original *Lemonade Stand* program of ten years ago. However, creating a simulation from scratch can teach students much more than using off-the-shelf software.

Computers are excellent tools with which to simulate real events: variables can be controlled and altered at will and events can occur faster than in real time. After trying various simulation programs, I found myself wondering how my sixth grade students and I might create such a program with LogoWriter. Our sixth grade class spent a week in the New England countryside with Nature's Classroom in the fall, and came back to school with some knowledge of predator-prey interactions. They continued their studies in science class. When I suggested to the students that they try to create their own predator-prey simulations in LogoWriter, they jumped at the chance.

LogoWriter, with four turtles and the ease with which it allows animation to be coded, is an excellent workbench for creating simulations. Before my students began work on their own simulations, I programmed a more complex version. This fulfilled two purposes. First, working on a complex Logo problem gives me great enjoyment. Second, whenever I assign a long-term complex task to my students, I try produce a simple version first so I can anticipate problems and guide them when necessary. In this case, I enjoyed the task so much that I created a much more complex version than usual.

With student projects, I usually strive to have the entire class have the same underlying program structure. Upon this foundation each student can add his/her own modifications. Throughout the project, I have stressed that the worlds the students created are their own, and may follow any rules, as long as a consistent logic holds true. Most solutions have come from the students, though at times I have given hints and prodded them towards a specific solution when too much time passed without a workable answer to a problem. The duration of this unit was one semester, with one or two class meetings per week.

The Setting

Students at St. Hilda's and St. Hugh's School practice keyboarding and use educational software in Nursery through first grade with their classroom teachers. Beginning in second grade, children also work with LogoWriter in the computer lab. The students meet with me once or twice a week for programming projects. Each class is forty minutes long. Class size varied this year from nine to twenty students. I usually begin each unit by introducing a new command or concept, or by posing a question or problem. Then the students work on experimentation, application, and testing throughout the duration of the unit, which can vary from one period to as long as one semester. Often, there are a few interrelated concepts and questions being worked on simultaneously. Units this year included word processing and book-making in the second grade, animated sentences in the third grade, coordinate plotting of designs in the fourth grade, ancient

Egypt-based projects in the fifth grade, derivation of Pi and the creation of pie graphs in the sixth grade, and independent programming projects (branching stories, animation, music 'videos', and multiple choice Latin exams) in the seventh and eighth grades. Many classes also use the lab for Writer's Workshops using LogoWriter or Word Perfect with their language arts teachers. The lab currently contains twenty-one MS/DOS computers (ranging from monochrome XTs with no hard drives to PS/2s) running LogoWriter 2.0.

Development of My Simulation

To lay a foundation for this project, I decided on the parameters for my ecosystem. My ecosystem would:

- be aquatic
- have one species of plant
- have one species of herbivorous fish
- have one species of carnivorous fish
- have two adults, male and female, of each fish species
- have the herbivores eat only plants
- have the carnivores eat only the herbivores
- have fish mate only with members of the same species
- have a random number and random distribution of plants each time the simulation was run

The number of species and the number of individuals per species were chosen for simplicity and because of LogoWriter's limitation of four turtles. I chose an aquatic ecosystem so my scenery would be simple: with no horizon line, the entire screen could use one background color.(see Figure 1 for a screen print-out) I next began creating the shapes needed for the ecosystem.(shape 2 was later removed) Shape numbers 1 through 8 were used (see Table 1 and Figures 2 - 8). Even-numbered turtles (0 & 2) were used for the herbivores and odd-numbered turtles (1 & 3) were used for the carnivores.(see Table 2) I began programming this project on a four-color CGA IBM/PC, so I decided to begin with three screen colors for my three species. I chose white for herbivores, cyan for plants, and magenta for carnivores. Other colors were added later for other platforms.(see Table 3a)
I began with a **setup** procedure to clear the screen:

```
to setup
rg
end
```

I then wrote a **set.fish** procedure to give the four turtles their proper shapes, colors, and headings:

```
to set.fish
tell all st pu home
tell 0 setsh 3
tell 1 setsh 6
tell 2 setsh 4
tell 3 setsh 7
tell [0 2] setc 1 seth 90
tell [1 3] setc 3 seth 270
end
```

The heading depended on whether the fish was facing left (**seth 270**) or facing right (**seth 90**).

However, before the fish begin to move, the plants should be stamped on the screen.

```
to set.grass
setsh 1 setc 2
end

to grow.grass
repeat 25 + random 25 [plant.grass]
end

to plant.grass
pu
setpos list random 320 ((-1 * random 94) + 8)
pd
stamp
end
```

Set.grass sets the shape and color of the turtle. **Grow.grass** calls the **plant.grass** procedure between 25 and 49 times, randomly. **Random** was used to give the simulation some variation each time it ran. The range was chosen to have sufficient plants for feeding, but not so many that the screen was mostly stamped plants. The **plant.grass** procedure places the turtle at a random position and stamps it. **Setpos** requires a **list** of two numbers as input. **Random 320** was chosen as the first, or "x" value, because this is the width of an IBM screen. For the "y" value, I wanted plants only in the lower half of the screen. Since the IBM screen is 189 pixels high, I chose one-half of that, or 94. By multiplying this by -1, only points in the lower half of the screen will be addressed. However, since shapes are 16 pixels high (in IBM, Apple IIGS, C64), I added half of this, or 8, to the number so the plants wouldn't be split between the bottom and top of the screen.

I was now ready to animate my fish with a **begin** procedure, using **herb** and **carn**:

```
to herb
fd random 10
end
```

```
to carn
fd random 10
end

to begin
tell 0 herb
tell 1 carn
tell 2 herb
tell 3 carn
begin
end
```

The **begin** procedure caused the four turtles to move across the screen one at a time anywhere from 0 to 9 steps each time by using **forward** with a **random 10**. **Begin** then calls itself, and continues until a **stop** command is entered from the keyboard.

Because the headings had been set at either 90deg. or 270deg., the fish would only move horizontally. Just as I had used **random** for **fd**, I could also use it for **heading**. I modified the **herb** and **carn** procedures:

```
to herb
fd random 10 seth 248 + random 45
end

to carn
fd random 10 seth 68 + random 45
end
```

Because of the way I had designed the shapes, the carnivores had to face right, and the herbivores left. By using **random**, the **heading** could vary but still face in the correct general direction. I chose a range of 44 degrees by using **random 45** which reports a range of 0 to 44. To face right, **68 + random 45** was used. This reports a range of 68 to 112, with an average of 90. To face left, **248 + random 45** was used. This reports a range of 248 to 292, with an average of 270.
With this enhancement, herbivores and carnivores moved around the screen with **random** distances and headings, colliding with each other randomly with a satisfactory frequency. I now needed to program feeding and mating.

```
to herb
fd random 10
seth 248 + random 45
if equal? colorunder 2 [feed]
if equal? colorunder 1 [herb.mate]
end
```

```
to carn
fd random 10
seth 68 + random 45
if equal? colorunder 1 [feed]
if equal? colorunder 3 [carn.mate]
end
```

Now, as the animals moved around the screen, the program tested for **colorunder**. Whenever a carnivore passed over **color 1** (an herbivore), **feed** would be called, and whenever it passed over **color 3** (the other carnivore), **carn.mate** would be called. For an herbivore, **feed** would be called whenever it passed over **color 2** (a plant), and **herb.mate** would be called whenever it passed over **color 1** (the other herbivore).

Feed was simple to program:

```
to feed
pe fill pu
end
```

Feed erases as much of the food shape as can be filled in, depending on the location and "contiguousness" of the pixels.

The two mating procedures work by first setting the shape of the female to the baby's shape, **stamp**ing this shape to the screen, then setting the shape back to the mother's. The **fd 20** commands, which produce a "rebound" effect, were added because multiple matings were occurring as adults of the same species remained in close proximity.

```
to herb.mate
tell 2 setsh 5
pd stamp pu
setsh 4 fd 20
end

to carn.mate
tell 3 setsh 8
pd stamp pu
setsh 7 fd 20
end
```

Some problems arose at this point. While herbivores could feed on plants by erasing them, a carnivore feeding on an adult had no effect on it because a turtle cannot be erased. In addition, once babies began to be stamped on the screen, adults would mate as they passed over their offspring. I also discovered at this point that in Macintosh and Commodore 64 LogoWriter, while **colorunder** will test for colors stamped, filled, or drawn on the screen, it will not test for turtle color. Another test had to be used. One solution is to test for the distance between the male and female.(See the end of the program listing for an example.) The program at this point was as follows:

```
to startup
setup
set.grass
grow.grass
set.fish
begin
end
```

I had successfully written a simple simulation. I was really quite excited seeing my fish move around the screen, feeding and stamping babies! While this was a good start, I wanted to go further. I drew up a list of features I wanted. Many of these were arbitrary, but I decided that I could create any rules for my ecosystem as long as there was a consistent logic applied:

- The simulation should not be tied to any one platform running LogoWriter
- The number of feedings should be tallied
- The number of births should be tallied
- The two species should give birth in different areas of the screen to reinforce their niche differences
- A timer should be added, so feeding and mating rates could be calculated and compared
- Grass should continue to grow throughout the simulation
- Fish should not wrap around the screen vertically because it looks odd
- Adult and offspring should be different colors to avoid intergenerational mating
- Carnivores should eat herbivore babies, but not herbivore adults
- Carnivores should eat their own offspring some of the time (life is tough)
- Females should be able to reproduce parthenogenically after having fed a specified number of times
- Proximity, and not COLORUNDER, must be used for mating to allow for Macintosh and C64 compatibility
- Species-specific courtship rituals should be added

These features were added to the simulation.(See annotated program listing in Addendum A. A flow chart is shown in Figure 9)
While this simulation can now run on all versions of LogoWriter, be aware that it will be quite slow when running on anything slower than an IBM 286. For Apple IIGS LogoEnsemble users, type **newpage40** to set 40 columns before you name your page. Because the Commodore 64 has very little available memory and because only LogoWriter 1.1 is available for it, many deletions and modifications have to made.(See Addendum D)

Development of a Student Simulation

To lay a foundation for the student project, I asked each student to come up with a set of rules for ecosystems in general. They merged these into a combined list, then derived a shorter list of general rules from these during a brainstorming session. The next step was to divide students into cooperative groups of four or five students each. Each group had to decide what type of ecosystem they wanted, and to state rules for their ecosystem. I created 3 groups of four boys and one group of five girls.

For simplicity, each ecosystem was allowed only one predator species, one prey species, and one plant species. Students then began creating the shapes needed for their ecosystem.(See Table 1 and Figures 1-8.) For class-wide consistency and easier teacher debugging, I suggested using even-numbered turtles (0 & 2) for the herbivores, odd-numbered turtles (1 & 3) for the carnivores (See Table 2) white for herbivores, cyan for plants, and magenta for carnivores. Other colors, if available, were used for scenery and background.(See Table 3b) We began with a **setup** procedure to give the four turtles their proper shapes, colors, and headings:

```
to setup
tell all st pu home
tell [0 2] setc 1 seth 90
tell [1 3] setc 3 seth 270
tell 0 setsh 3
tell 1 setsh 6
tell 2 setsh 4
tell 3 setsh 7
end
```

The heading depended on whether the animal was facing left (**seth 270**) or facing right (**seth 90**).

The next step was to create scenery for the ecosystem. Students wrote procedures to stamp their plants on the screen. For terrestrial ecosystems, students split the screen with a horizon line. Other components, such as ponds, mountains, etc., were added by some students. A typical **scenery** procedure is shown below: (See Addendum F for full program listing)

```
to scenery
scene
pond
trees
sun
end
```

Students then spent a few weeks creating scenarios, such as a giraffe walking up to a tree and eating from its crown, or a wolf chasing a deer. The girls initially concentrated on mating and giving birth, whereas the boys had concentrated on predation. With reporters from each group often seeing what others were doing, the girls added predation and the boys added mating. My reason for creating an all-girl group had been to allow for full female expression. This was successful and had affected the boys as I had hoped. After all groups completed this stage, the students watched each other's simulations. They realized that they had created plays with actors following set lines, rather than free-flowing and unpredictable simulations. I suggested that it might be possible to move the animals around the screen randomly so that interactions could not be predicted. We wrote a **move** procedure, using **herb** and **carn**:

```
to herb
fd random 10
end
```

```
to carn
fd random 10
end

to move
tell 0 herb
tell 1 carn
tell 2 herb
tell 3 carn
move
end
```

The move procedure caused the four turtles to move across the screen anywhere from 0 to 9 steps each time by using **forward** with a **random 10**.

Next, students wanted to be able to have their animals eat, so I introduced **colorunder**. They modified the **herb** and **carn** procedures as follows:

```
to herb
fd random 10
if equal? colorunder 2 [eat]
end

to carn
fd random 10
if equal? colorunder 1 [eat]
end
```

As in my program, as the animals moved around the screen, the program tested for **colorunder**. If the carnivore passed over color 1 (an herbivore), it would eat, while an herbivore would eat whenever it passed over color 2 (a plant). It was now up to the students to devise an eat procedure. In a class discussion, three possibilities were suggested:

```
to eat
pe stamp pu
end
```
This takes a chunk out of the food the size and shape of the eater. One student likened it to a cartoon where a character runs through a wall and leaves a hole its size and shape.

```
to eat
pe fill pu
end
```
This erases as much of the food shape as can be filled in depending on the location and contiguousness of the pixels.

```
to eat
pe fd 1 pu
end
```
This takes a one-pixel bite out of the food.

Students decided which of these three they wanted to use. Most chose to use **fill**. They realized that when the carnivore passes over an herbivore, it won't affect it because while a stamped **shape** can be erased, a turtle cannot. Students then decided to have their animals give birth so that **stamp**ed babies could be eaten. To tackle the programming to control mating, **colorunder** was used again. Since the students were working in MSDOS LogoWriter, **colorunder** does test for turtle color. This was added to **herb** and **carn**:

```
to herb
fd random 10
if equal? colorunder 2 [eat]
if equal? colorunder 1 [h.mate]
end

to carn
fd random 10
if equal? colorunder 1 [eat]
if equal? colorunder 3 [c.mate]
end
```

Now, if one carnivore passed over another, it would call the **c.mate** procedure. Likewise, if one herbivore passed over another, it would call the **h.mate** procedure.

```
to h.mate
tell 2 setsh 5
pd stamp pu
setsh 4 fd 20
end

to c.mate
tell 3 setsh 8
pd stamp pu
setsh 7 fd 20
end
```

These procedures work by first setting the **shape** of the female to the baby's **shape**, **stamp**ing this **shape** to the screen, then setting the **shape** back to the mother's. Students added the **fd 20** commands, to avoid the multiple matings that occurred as adults of the same species remained in close proximity.

In some cases, no matings occurred because males and females of the same species would never collide during a simulation run. Students saw that if animals were at different places vertically on the screen and only moved horizontally, they would never meet. Some students then realized that

just as they had used **random** for **fd**, they could also use it for **heading**. Again, they modified the **herb** and **carn** procedures:

```
to carn
fd random 10
seth 248 + random 45
if equal? colorunder 1 [eat]
if equal? colorunder 3 [c.mate]
end

to herb
fd random 10
seth 68 + random 45
if equal? colorunder 2 [eat]
if equal? colorunder 1 [h.mate]
end
```

Students decided that the heading should vary by a small range, because when larger values were used, the animal appeared to be jumping around the screen. Most students chose my suggested "window" of 45 degrees.

Students did not like to see their animals wrapping around the screen top and bottom, so two final lines were added to both **herb** and **carn**:

```
if ycor > 0 [descend]
if ycor < -80 [ ascend]
```

The minimum and maximum values for **ycor**, the "y" coordinate, or vertical location, depend on what part of the screen a student wanted to allow the animals to roam in. These values varied for each group.

Descend and **ascend** simply moved the turtle in question up or down:

```
to descend
seth 180
repeat 20 [fd 1]
end

to ascend
seth 0
repeat 20 [fd 1]
end
```

Personal Successes

- I found myself forced to tackle some concepts which I had previously avoided. Naming variables by using **word** and the number of the turtle in question:

  ```
  make word "eatcnt who 1 + thing word "eatcnt who
  ```

  allowed me to write shorter code which was then applicable to all four turtles, rather than writing four separate procedures.

- The frustrations of creating a procedure that would run under eight different platforms (IBM- CGA, IBM-EGA, IBM-VGA, Apple II, Apple IIGS, LogoEnsemble, Commodore 64, and Macintosh) forced me to use variables for screen sizes and colors in ways I never would have otherwise.
- The problem of **colorunder** not working on the Macintosh the same way as in other platforms forced me to work out the Pythagorean Theorem in LogoWriter (with twenty parentheses!)
- The problem of **colorunder** not working on the Commodore 64 the same way as in other platforms coupled with a lack of the **sqrt** function forced me to work out a different distance test, based on absolute value, in LogoWriter.
- As I was working on this for my own enjoyment, my sixth graders were working on similar simulations. I was able to anticipate some problems, and guide them more efficiently.

Successes in the Classroom

This project was largely student-designed and directed, and created more interest than almost any other project and class I've worked with. New concepts, such as **colorunder** and **random**, were introduced as they were needed. Student understanding and application of these concepts was rapid and flexible. Previously learned concepts, such as third grade animation, heading, multiple turtles, and turtle interactions have been reinforced and expanded upon. Students were able to work in groups, individually, and as a class during this project. Science and classroom teachers stopped by to see our progress.

Problems with Program Execution

- With CGA graphics on the IBM PC, only three pencolors are possible. Because of this, animals "mate" with their offspring, and feeding is counted when carnivores pass over herbivores, even though they have no effect on their prey. On any other platform, at least 5 colors would be available, and offspring could be stamped a different color than their parents.
- LogoWriter is slow to begin with. This simulation, with up to seven levels of sub-procedures, runs slowly even on a fast IBM. I have used *Apple File Exchange* to pull this simulation as a text file into the Macintosh, the Apple IIe, and the Apple IIGS where I could tinker with it. On Apples and on slower Macintoshes, it is very slow. There is no

loadtext command for Commodore 64 LogoWriter 1.1, so I had to retype it. Because of memory limitations, extensive modifications had to made for it to run on the C64.(See Addendum D) Even then, it is painfully slow at 1 MHz.

Problems in the Classroom

As with any semester-long project, it is sometimes difficult to maintain a high level of student interest. The main obstacle for most students was the limitation of three pencolors with CGA graphics. Some students found parent-offspring matings disturbing. I challenged the students to devise solutions to this problem. Laura, who was working with 16-color EGA, chose to use different colors for the offspring. Austin stamped his babies on the bottom of screen where the parents didn't go. I did not have time to introduce a distance test, rather than **colorunder**, and no student thought of it during our project.

Proposed Experimentation and Additions:
Using the Simulation as a Science/Math Lab Resource

This simulation can now be run a number of times to see the outcomes. Values of variables can be changed to see how that affects the outcome.

Some questions I have not yet investigated are:

- What happens if predator and prey move at greatly different speeds?
- Is there a way to measure population growth and overpopulation?
- Could animals be made to die as their food source disappears?
- Could waste or pollutants be made to build up during a simulation run? (addressed by one student, Austin)
- Could a species be made extinct if the ratio of feeding to time becomes unfavorable?
- Could disease or parasite infestation be made to increase as population size increases?

Those students who are interested will now have time to implement some of their suggestions.

Some questions I have looked into are:

- Within one simulation run:
    - Do males and females of the same species behave differently?
    - Do carnivores and herbivores mate at the same rate?
    - Could statistics such as the number of feedings and matings be kept as the program runs?
- Comparing different simulation runs:
    - Will the number of matings or feedings differ greatly each time the program is run?
    - What is the distribution pattern of the feeding or mating frequencies?

To collect data on these questions, I added a **stats** (statistics) procedure to my simulation:

```
to stats
ef
if member? "stats filelist [loadtext "stats] bottom
(pr :day :eatcnt0 :eatcnt2 :eatcnt1 :eatcnt3
:h.births int :eatcnt2 / 20
:c.births int :eatcnt3 / 20)
if member? "stats filelist [erasefile "stats]
savetext "stats
end
```

Stats first clears the text screen. If there is already a **stats** file on the disk, it is loaded. The cursor is moved to the bottom of the text screen. The values of the following variable are printed on the screen:

```
:day              the day in numbers
:eatcnt0          male herb feedings
:eatcnt2          female herb feedings
:eatcnt1          male carn feedings
:eatcnt3          female carn feedings
:h.births         births from herbivore matings
:int eatcnt2 / 20 parthenogenic herb births
c.births          births from carnivore matings
:int eatcnt3 / 20 parthenogenic carn births. The old stats
                  file is erased from the disk. The new stats
                  file with this day's data is saved.
```

This procedure was then added to **rollover**, so it would be called once a day at 12 midnight:

```
to rollover
stats
make "time 0
make "meridian "AM make "day :day + 1
end
```

The simulation was allowed to run overnight, until 521 simulated days' worth of data had been accumulated. These data were then graphed with *PFS First Graphics*. The male and female carnivore feeding rates stay approximately equal during the 521 day run.(See Figure 10) Females end up slightly higher at the end of the run. Male and female herbivore feeding rate stay equal until day 150, after which the male rate decreases. Since the movements around the screen are random, this raises interesting questions. Could the female rates be higher because they spend more time near the top (carnivore) or bottom (herbivore) of the screen, thus avoiding male competition? Or is it because the turtles are moving at the rate of **who + fd 5**, and the females,

with their higher turtle numbers, move faster than the males? By making the **fd**s all the same, this question could be answered.

Herbivore and carnivore births were compared. (See Figure 11) Here, equal rates were expected for births dues to matings (from random collisions), but higher parthenogenic rates were expected for herbivores, since they feed much more often than carnivores. Surprisingly, the carnivore birth rate (1391 births/521 days, or approximately 2 2/3 births/day) was much higher than the herbivore birth rate (873 births/521 days, or approximately 1 2/3 births/day). I am at a loss to explain these differences.

In addition to analyzing summed data, numerous daily data can be collected and compared. I changed **rollover** to:

```
to rollover
stats
clearnames
start
end
```

This will allow the simulation to run for only one day before all variable values are cleared and the program starts again. I let it run for 112 daily cycles.

I then divided the feedings per day into groups of ten for the herbivores and groups of three for the carnivores so I could graph the data discretely on a histogram rather than continuously with a line graph. I counted the feeding frequencies for males and females of both species. For example, there were 0 days when the herbivore females fed 0-10 times, 5 days when they fed 11- 20 times, 20 days when they fed 41-50 times, etc. (See Figure 12 for herbivores, Figure 13 for carnivores) All four approach normal distributions. Possibly with a larger sample, a smoother normal distribution could be seen.

This type of data collection and analysis could be an excellent way to introduce students to normal distribution, to statistics, and to use of the computer as a math/science laboratory.

Addendum A

Ecology Simulation Program Listing

```
to start
setup
set.grass
grow.grass
set.fish
begin
end

to ef
if not front? [flip]
ct
end
```

Ct (cleartext) is dangerous to use because if it is used in the command center of your flip side, ALL text (i.e., all procedures) will be erased. I urge you to put this **E**rase**F**ront procedure on each of your pages, and use **ef** instead of **ct**. This procedure will clear the text only after you have flipped to the front side.

```
to setup
rg
ht
ef
cc
make "width 320
make "height 190
make "eatcnt0 0
make "eatcnt1 0
make "eatcnt2 0
make "eatcnt3 0
make "day 1
make "time 0
make "meridian "AM
make "herb.m.s 3
make "herb.f.s 4
make "herb.b.s 5
make "carn.m.s 6
make "carn.f.s 7
make "carn.b.s 8
make "plant.sh 1
make "h.births 0
make "c.births 0
make "male "M
make "female "F      See Addenda B - E for additions or deletions here.
setbg :water.5       Keep this as the last line in this procedure even
end                        as other lines are added.
```

The **setup** procedure creates a number of global variables used throughout this program:

:width	is the width of the screen in pixels.
:height	is the height of the screen in pixels.
:eatcnt0-3	will keep a count of how many times turtles 0-3 have
:day	eaten.
:time	is first set to 1.
:meridian	is first set to 0.
:herb.m.s	is first set to AM.
:herb.f.s	is the herbivore male shape number.
:herb.b.s	is the herbivore female shape number.
:carn.m.s	is the herbivore baby shape number.
:carn.f.s	is the carnivore male shape number.
:carn.b.s	is the carnivore female shape number.
:plant.sh	is the carnivore baby shape number.
:h.births	is the plant shape number.
:c.births	will keep a count of herbivore births.

:male	will keep a count of carnivore births.
:female	is set equal to the letter M.
	is set equal to the letter F.

Modifications to **Setup** for Different Platforms

Most of these lines set the colors of the fish, plants, and water.(See Table 3a) Other lines are explained in the addenda.(**IBM** users see Addendum B. **Apple** users see Addendum C. **Commodore 64** users see Addendum D. **Macintosh** users see Addendum E.)

```
to set.grass
setsh :plant.sh
setc :plant.color
end
```

Sets the turtle to the plant's **shape** and **color**.

```
to grow.grass
repeat 25 + random 25 [plant.grass]
end


to plant.grass
pu setpos list
random :width ((-1 * random (:height / 2)) + 8)
pd stamp
end
```

Sets the position of the turtle anywhere along the width of the screen, and anywhere in the bottom half of the screen.

```
to set.fish
tell all st
pu home
tell 0 setsh :herb.m.s
tell 1 setsh :carn.m.s
tell 2 setsh :herb.f.s
tell 3 setsh :carn.f.s
tell [0 2] setc :herb.color
tell [1 3] setc :carn.color
end
```

Sets the four turtles to their proper shapes and colors.

```
to begin
tell 0 herb
tell 1 carn
tell 2 herb
tell 3 carn
circadian
begin
end
```

Moves each turtle alternately, updates the timer (see **circadian**), then calls itself recursively.

```
to herb
seth 248 + random 45
fd who + random 5
if equal? colorunder :plant.color [feed eat.count]
ask 0 [make "pos0 pos]
ask 2 [make "pos2 pos]
if ((hypotenuse :pos0 :pos2) < 16) [herb.mate give.birth]
swim.test
end
```

The heading is set to the left, anywhere from 248 to 292. The turtle moves forward a number of steps equal to random 5 plus its turtle number. **Who** was added to the random so that males and females would move at different speeds. If the **colorunder** is equal to the plant color, **feed** is called, then **eat.count** (see **eat.count**). The **position**s of the male and female are stored in variables and these values are passed to **hypotenuse**. If the distance between the two turtles is less than 16 steps, **herb.mate** and **give.birth** are called. This distance can be changed to a different value depending on how "attracted" you want your males and females to be to each other. See **swim.test** below.

```
to carn
seth 68 + random 45
fd who + random 5
if equal? colorunder :herb.baby [feed eat.count]
if and (equal? colorunder :carn.baby)
(equal? 0 (random 2)) [feed eat.count]
ask 1 [make "pos1 pos]
ask 3 [make "pos3 pos]
if ((hypotenuse :pos1 :pos3) < 16) [carn.mate give.birth]
swim.test
end
```

This procedure is identical to **herb** with the following exceptions:

- The heading is set to the right, from 68 to 112.
- If the **colorunder** is equal to the herbivore baby color, **feed** is called, then **eat.count** (see **eat.count**).

- If the **colorunder** is equal to the carnivore baby color, **feed** and **eat.count** are called randomly about 50% of the time (see **eat.count**).
- **Position** is used identically, but with different variable names. See **swim.test**.

```
to swim.test
if ycor > :height / 2 - 24 [seth 180 swim]
if ycor < 8 - :height / 2 [seth 0 swim]
end
```

If the turtle is within 24 steps of the top of the screen, the **heading** is changed to **180** and **swim** is called. If the turtle is within 8 steps of the bottom of the screen, the **heading** is changed to **0** and **swim** is called. This keeps the turtle from wrapping around the screen top and bottom.

```
to swim
repeat :height / 2 [fd 1]
end
```

The turtle moves away from where it is a distance of half the height of the screen.

```
to eat.count
cc
make word "eatcnt who 1 + thing word "eatcnt who
if member? who [2 3] [check.feedings] type [The number of
feedings is:]
t13
(type :male [herb=] :eatcnt0 [;] :female [herb =] :eatcnt2)
t13
(type :male [carn=] :eatcnt1 [;] :female [carn =] :eatcnt3)
t13
(type [herb births =] :h.births [; carn. births =] :c.births)
end
```

This procedure tallies feedings and displays the current values.

```
make word "eatcnt who 1 + thing word "eatcnt who
```

This works by adding 1 to the current value (thing) of the variable called **:eatcnt0**, **:eatcnt1**, **:eatcnt2**, or **:eatcnt3**. Which **eatcnt** variable is used is determined by making a **word** out of **eacnt** plus **who**, the number of the turtle which called this procedure. If the turtle in question is female [2 3], **check.feedings** is called.

The current values of the numbers of feedings for each turtle and the number of matings for each species are displayed in the command center.

```
to check.feedings
if equal? 0
(remainder thing word "eatcnt who 20)
[make word "eatcnt who 1 + thing word "eatcnt who labor]
end
```

Females will give birth parthenogenically every twenty times they feed. If the number of times they have fed divided by 20 gives a remainder of zero, **labor** is called.

```
to labor
ifelse equal? who 2 [seth 180 descend] [seth 0 ascend]
give.birth
end
```

If the turtle in question is 2, it will face down and **descend**, otherwise, it is turtle 3 and it will face up and **ascend**. **give.birth** is called.

```
to t13
type char 13
end
```

Moves the cursor to the next line in the command center.

```
to feed
pe fill pu
end
```

Feeding is accomplished by erasing what is under the turtle's pen. This can vary from one pixel to a large area, if the **color** is contiguous.

```
to give.birth
birth.check
ifelse equal? who 2
[setsh :herb.b.s setc :herb.baby pd stamp pu setsh :herb.f.s
setc :herb.color seth 270][setsh :carn.b.s setc :carn.baby pd
stamp pu setsh :carn.f.s setc :carn.color seth 90]
swim
end
```

See **birth.check**.

If the turtle is #2, the herbivore baby color and shape are set, and stamped. Otherwise, the carnivore baby color and shape are set and stamped. The shape and color of the female is restored. See **swim** above.

```
to birth.check
if member? colorunder
[:herb.baby :carn.baby]
[fd 9 birth.check]
end
```

If the **colorunder** the female is already that of her own baby, she will move 9 steps. This continues until a clear birthing area is found.

```
to circadian
ef
make "time :time + 1 make "hour (int :time / 60)
make "min remainder :time 60 if equal? :min 0 [water]
if :min < 10 [make "min word "0 :min]
make "display word (word :hour ":) :min
if equal? 0 (remainder :time 1440) [rollover]
(insert [It is day] word :day ", :display :meridian)
if equal? (random 50) (0) [random.grass]
end
```

This is a completely arbitrary timer. Possibly those with Macintoshes may want to use the clock built into the computer. Time is incremented by 1. The hour is equal to the integer value of **:time** divided by 60. The minutes are equal to the remainder of **:time** divided by 60. If **:min** are equal to zero (i.e., every hour on the hour), the **water** procedure is called. If **:min** is less than 10, a zero is added in front to pad the number. Whenever **:time** is equal to a multiple of 1440, **rollover** is called. The day and time are displayed. Approximately 1 out of 50 times (2%) one grass will be stamped with **random.grass**.

```
to water
if equal? :hour 6
[setbg :water.1] if AND :hour > 6 :hour < 12
[setbg :water.2] if equal? :hour 12
[setbg :water.3 make "meridian "NOON]
if :hour > 12
[make "meridian "PM]
if equal? :hour 18
[setbg :water.4] if :hour > 18
[setbg :water.5]
end
```

The background color is changed at 6 am, 12 noon, 1 pm, 6 pm, and 7 pm (for the APPLE II, with its 6 colors, all values of **:water.?** are black).

```
to random.grass
make "pos pos
make "shape shape
make "color color
set.grass
plant.grass
setsh :shape
setc :color
pu
setpos :pos
end
```

This procedure will randomly plant grass on the screen. Because all four turtles are already being used for the four fish, the computer must first store the position, shape, and color of the turtle in question. After one grass has been stamped, the position, shape, and color are restored. This happens so fast that only a momentary blinking is seen.

```
to rollover
make "time 0
make "meridian "AM
make "day :day + 1
end
```

This will reset the time to 0, and increment the day by 1.

```
to herb.mate
tell [0 2] repeat 12 [fd 2 rt 30]
repeat 12 [fd 2 lt 30]
tell 2 seth 180
descend
make "h.births :h.births + 1
end

to carn.mate
tell [1 3]
seth 45
repeat 8 [fd 5 rt 90 fd 5 lt 90]
repeat 8 [bk 5 rt 90 bk 5 lt 90]
tell 3 seth 0
ascend
make "c.births :c.births + 1
end
```

These two procedures are the different courtship dances performed by these two species. The number of births is incremented by 1.

```
to descend
fd 1
if ycor < 10 + (random 10) - :height / 2 [stop]
descend
end
```

The turtle will keep moving until it is within 10 to 19 steps of the bottom of the screen.

```
to ascend
fd 1
if ycor > :height / 2 - 24 - random 24 [stop]
ascend
end
```

The turtle will keep moving until it is within 24 to 47 steps of the top of the screen.

```
to hypotenuse :a :b
op (sqrt ((sq ((first :a) - (first :b))) + (sq ((last :a) -
(last :b)))))
end
```

The inputs to **hypotenuse** are the position of the two turtles in question. **Pos**ition is a list of two numbers (**xcor** and **ycor**). **Hypotenuse** will report the straight line distance between the two turtles by using the Pythagorean Theorem. The **first** of a **pos**ition in its **xcor**, and the **last** is its **ycor**. The squared differences in the **xcor**s are added to the squared differences of the **ycor**s, and the square root of this sum is output.

```
to sq :num
op :num * :num
end
```

Outputs the square of a number. Used in **hypotenuse**.

Or

I needed an alternative for the Commodore 64, which does not have **sqrt** in its LogoWriter 1.1. It was also suggested to me that a simpler measure of distance be used so that younger students will understand the procedure (summing the difference in Xs and Ys - a measure known as Manhattan Grid Distance). My solution was to sum the absolute values of the difference in "x" values and the difference in "y" values. This will work for any version of LogoWriter.

```
to hypotenuse :a :b
op (abs ((first :a) - (first :b))) + (abs ((last :a) - (last
:b))))
end
```

```
to abs :num
ifelse :num < 0 [op -1 * :num] [op :num]
end
```

You can replace the above **hypotenuse** procedure with this, but realize that it will no longer calculate the straight-line distance between two points. It might be a good idea to change its name here and in **herb** and **carn** where it is called.

Addendum B

For IBM PC LogoWriter Users

- Add these lines to **setup**

```
getshapes
make "carn.color 3
make "herb.color 1
make "plant.color 2
```

- Replace these lines in **setup**

```
make "male char 11
make "female char 12
```

These replace **m** and **f** with the male and female signs.

- Add these lines to **setup** if you have EGA(16 color) or VGA(256 color) graphics

```
make "carn.baby 11
make "herb.baby 9
```

- Add these lines to **setup** if you have CGA(4 color) graphics

```
make "carn.baby 3
make "herb.baby 1
```

Because there are only four pen colors with CGA, the offspring will be the same colors as their parents. This will lead to higher mating and feeding counts.

- Add these lines to **setup** if you have CGA or EGA graphics

```
make "water.1 6
make "water.2 10
make "water.3 13
make "water.4 5
make "water.5 0
```

- Add these lines to SETUP if you have VGA graphics

```
make "water.1 18
make "water.2 17
make "water.3 16
make "water.4 5
make "water.5 207
```

More shades of blue are available with VGA colors.

Addendum C
For the Apple II

For Apple II LogoWriter Users

- Add these lines to **setup**

```
getshapes
make "carn.color 3
make "carn.baby 4
make "herb.color 1
make "herb.baby 5
make "plant.color 2
make "water.1 0
make "water.2 0
make "water.3 0
make "water.4 0
make "water.5 0
```

- Replace these lines in **setup**

```
make "width 280
make "height 180
```

The screen heights and widths are smaller for the Apple II.

For Apple IIGS (with or without Ensemble) LogoWriter Users

- Add these lines to **setup**

```
make "carn.color 3
make "carn.baby 11
make "herb.color 1
make "herb.baby 9
make "plant.color 2
make "water.1 6
make "water.2 10
make "water.3 13
make "water.4 5
make "water.5 0
```

- Replace these lines in **setup**

```
make "male char 11
make "female char 12
```

These replace **m** and **f** with the male and female symbols.

For Apple IlGS Non-Ensemble LogoWriter users

- Add this line to **setup**

```
getshapes
```

Addendum D

For Commodore 64 LogoWriter Users

- Do not type in the following procedures

```
eat.count
check.feedings
circadian water
rollover
sq :num
hypotenuse
```

- Add these lines to **setup**

```
make "carn.color 2
make "carn.baby 10
make "herb.color 1
make "herb.baby 15
make "plant.color 3
```

- Delete these lines from **setup**

```
make "eatcnt0 0
make "eatcnt1 0
make "eatcnt2 0
make "eatcnt3 0
make "day 1
make "time 0
make "meridian "AM
make "h.births 0
make "c.births 0
make "male "M
make "female "F
setbg :water.5
```

- Delete this command from **begin**: **circadian**
- Delete this command from **herb**: **eat.count**
- Delete this command found in two places in **carn**: **eat.count**
- Delete this line from **herb.mate**: **make "h.births :h.births + 1**
- Delete this line from **carn.mate**: **make "c.births :c.births + 1**
- Add these procedures:

```
to xcor
op first pos
end
```

```
to ycor
op last pos
end
```

Use the second hypotenuse (Manhattan Grid Distance) procedure. There is no **sqrt** command in the Commodore 64's LogoWriter 1.1.

```
to abs :num
ifelse :num < 0 [op -1 * :num] [op :num]
end

to hypotenuse :a :b
op (abs ((first :a) - (first :b))) + (abs ((last :a) - (last
:b))))
end
```

Addendum E

For Macintosh LogoWriter Users

- Add these lines to **setup**

```
make "carn.color 6
make "carn.baby 37
make "herb.color 8
make "herb.baby 9
make "plant.color 3
make "water.1 171
make "water.2 188
make "water.3 208
make "water.4 242
make "water.5 180
```

- Replace these lines in **setup**

```
make "width 495
make "height 220
```

The screen heights and widths are larger for the Macintosh.

Table 1: Shapes Used

Shape Variable	Shape	Object Category	My Simulation	Student Simulation
:plant.sh	1	plant	grass	tree
:herb.m.s	3	male herbivore	fish	giraffe
:herb.f.s	4	female herbivore	fish	giraffe
:herb.b.s	5	baby herbivore	fish	giraffe
:carn.m.s	6	male carnivore	fish	lion
:carn.f.s	7	female carnivore	fish	lioness
:carn.b.s	8	baby carnivore	fish	lion cub

Table 2: Turtle Identities

Turtle Number	Identity
0	male herbivore
1	male carnivore
2	female herbivore
3	female carnivore

Table 3a: Colors Used in my Simulation

Color-Variable	IBM-CGA	IBM-EGA	IBM-VGA	Macintosh	Apple-IIGS	AppleII	C 64
		magenta-3	magenta-3	magenta-6	magenta-3	purple-3	red-2
:carn.color	magenta-3	white-1	white -1	lt gray-8	white-1	white-1	white-1
:herb.color	white-1	cyan-2	cyan-2	green-3	cyan-2	green-2	cyan-3
:plant.color	cyan-2	lt magenta-11	lt magenta-11	lt magenta-37	ltmagenta-11	orange-4	pink-10
:carn.baby	magenta-3	t gray-9	lt gray-9	gray-9	lt gray-9	blue-5	gray-15
:herb.baby	white-1	brown-6	lavender-18	lt blue-171	brown-6	black-0	
:water.1	brown-6	lt cyan-10	med. blue-17	bl-green-188	lt cyan-10	black-0	
:water.2	lt cyan-10	lt blue-13	lt blue-16	grn-blue-208	lt blue-13	black-0	
:water.3	lt blue-13	blue -5	blue-5	dark blue-242	blue-5	black-0	
:water.4	blue -5	black-0	dk purple-207	lt purple -180	black -0	black-0	
:water.5	black -0						

Table 3b: Colors Used in the Student Simulation

Color Number	Color
0	black
1	white
2	cyan
3	magenta
4	red
5	blue
15	light green

Addendum F

A Typical Student Program Listing

This is an edited version of the program written by The EcoGroup. (The EcoGroup: Sandy Cid, Vanessa Cook, Kristina Negron, Maritza Schaeffer, and Laura Stickney Class of 1999 St. Hilda's & St. Hugh's School) I have removed all of their predetermined scenarios for the sake of space and simplicity.

```
to startup
getshapes
scenery
setup
move
end

to scenery
scene
pond
trees
sun
end

to scene
rg
setbg 15
setc 5
rt 90
fd 320
pu
lt 90
fd 15
pd
fill
end

to pond
pu
home
bk 30
rt 90
fd 145
setsh 12
setc 5
repeat 6 [lt 60 fd 7 pd stamp pu]
end
```

```
to trees
pu
home
bk 45
rt 90
setsh 1
setc 2
repeat 12 [pd stamp pu fd 32]
fd 16
lt 90
bk 20
rt 90
repeat 12 [pd stamp pu fd 32]
end

to sun
pu home
rt 90
fd 120
setsh 12
setc 4
lt 90
fd 68
pd
stamp
end

to setup
tell all
st
pu
home
tell [0 2]
setc 1 seth 90
tell [1 3] setc 3
seth 270
tell 0 setsh 3
tell 1 setsh 6
tell 2 setsh 4
tell 3 setsh 7
end
```

```
to move
tell 0 herb
tell 1 carn
tell 2 herb
tell 3 carn
move
end

to carn
fd random 10
seth 248 + random 45
if equal? colorunder 1 [eat]
if equal? colorunder 3 [c.mate]
if ycor > 0 [descend]
if ycor < -80 [ ascend]
end

to herb
fd random 10
seth 68 + random 45
if equal? colorunder 2 [eat]
if equal? colorunder 1 [h.mate]
if ycor > 0 [descend]
if ycor < -80 [ ascend]
end

to eat
pe
fill
pu
end

to c.mate
tell 3
setsh 8
pd
stamp
pu
setsh 7
fd 20
end
```

```
to h.mate
tell 2
setsh 5
pd
stamp
pu
setsh 4
fd 20
end

to descend
seth 180
repeat 20 [fd 1]
end

to ascend
seth 0
repeat 20 [fd 1]
end
```

As stated earlier, if you want to test for distance rather than **colorunder** (you must on the Macintosh and Commodore 64), you will need to replace **if colorunder = 1 [h.mate]** in **herb** with:

```
ask 0 [make "pos0 pos]
ask 2 [make "pos2 pos]
if ((hypotenuse :pos0 :pos2) < 16) [h.mate]
```

In **carn** you will need to replace **if colorunder = 3 [c.mate]** with:

```
ask 1 [make "pos1 pos]
ask 3 [make "pos3 pos]
if ((hypotenuse :pos1 :pos3) < 16) [c.mate]
```

You will then need to add the **hypotenuse** and **sq** procedures for straight-line distance (not for the C64), or **hypotenuse** and **abs** for Manhattan Grid Distance, found on the last page of my program listing.

Figure 9: Ecology Simulation Flow Chart

```
start
    setup
    set.grass
    grow.grass
        plant.grass
    set.fish
    begin
    herb
        feed
        eat.count
            check.feedings
                labor
                    descend
                    give.birth
                        birth.check
                        swim
        hypotenuse
            sq
            herb.mate
            give.birth
                birth.check
                swim
        swim.test
            swim
    carn
        feed
        eat.count
            check.feedings
                labor
                    ascend
                    give.birth
                        birth.check
                        swim
        hypotenuse
            carn.mate
            give.birth
                birth.check
                swim
        swim.test
```

```
            swim
    circadian
        water
        rollover
        random.grass
            set.grass
            plant.grass
    begin
end
```

Figure 1: The screen after the simulation has run for a while.
Note herbivore and carnivore babies.

(Top: Teacher simulation - Bottom: Student simulation)

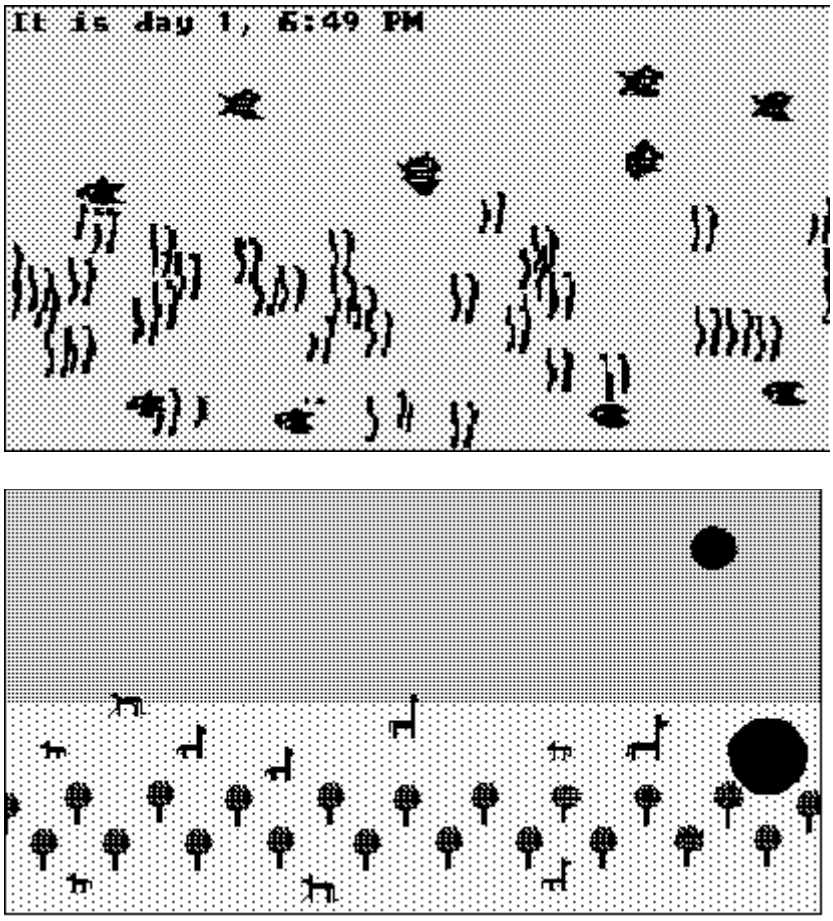


Figure 2: Shape #1 - Plant

(Top: Teacher simulation - Bottom: Student simulation)

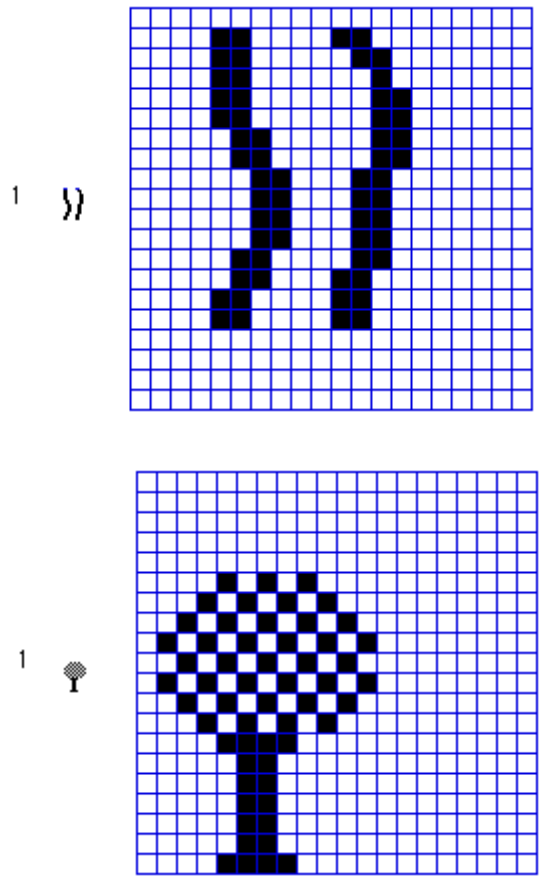
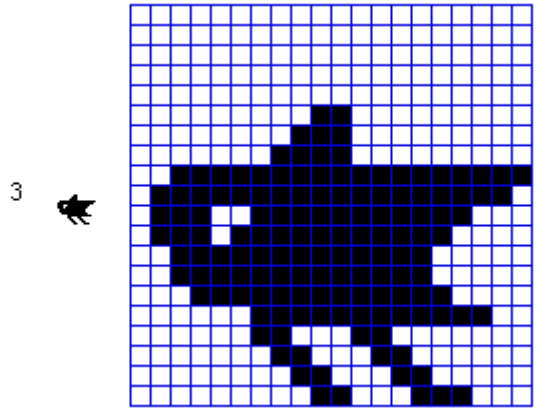


Figure 3: Shape #3 - male herbivore

(Top: Teacher simulation - Bottom: Student simulation)



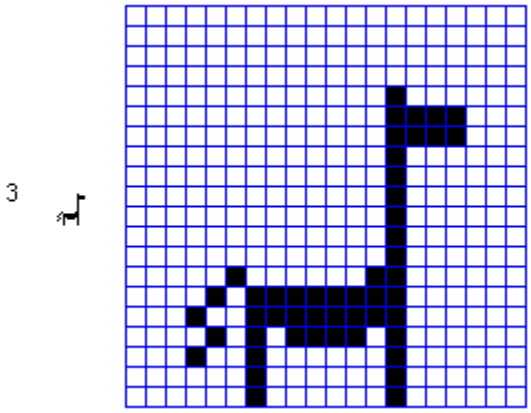


Figure 4: Shape #4 - female herbivore

(Top: Teacher simulation - Bottom: Student simulation)

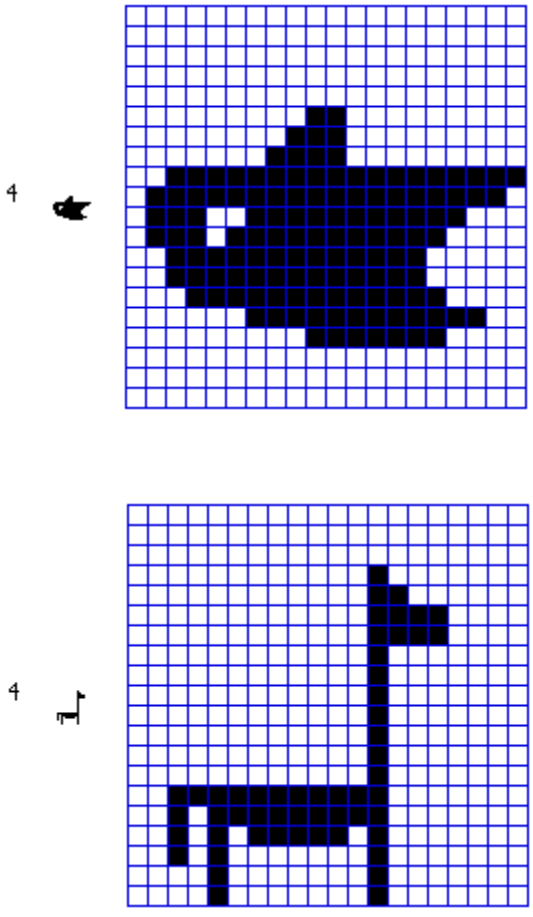


Figure 5: Shape #5 - baby herbivore

(Top: Teacher simulation - Bottom: Student simulation)

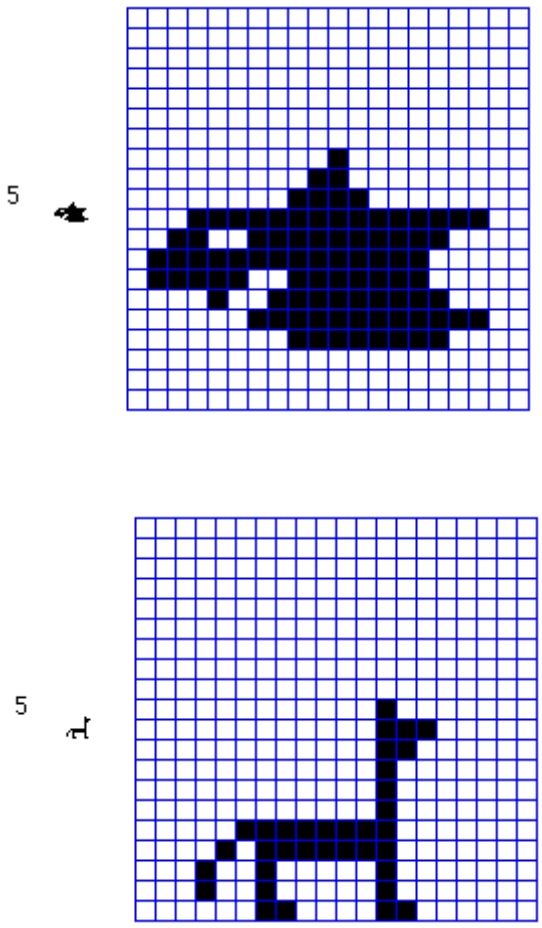


Figure 6: Shape #6 - male carnivore

(Top: Teacher simulation - Bottom: Student simulation)

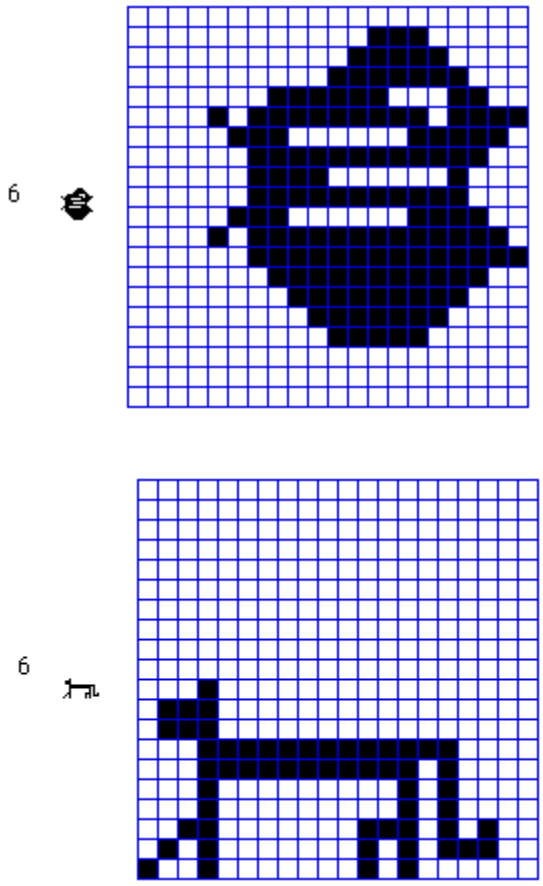
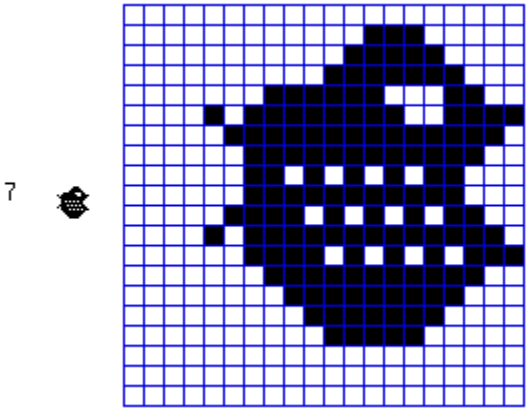


Figure 7: Shape #7 - female carnivore

(Top: Teacher simulation - Bottom: Student simulation)



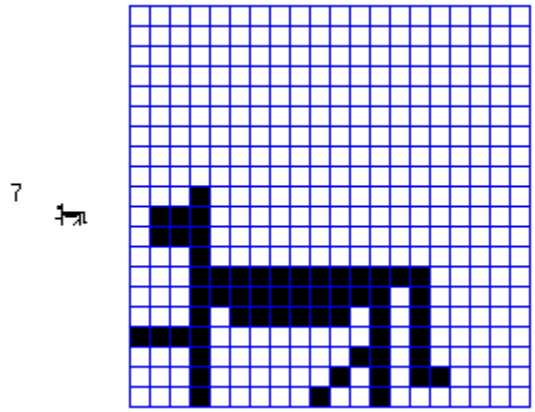
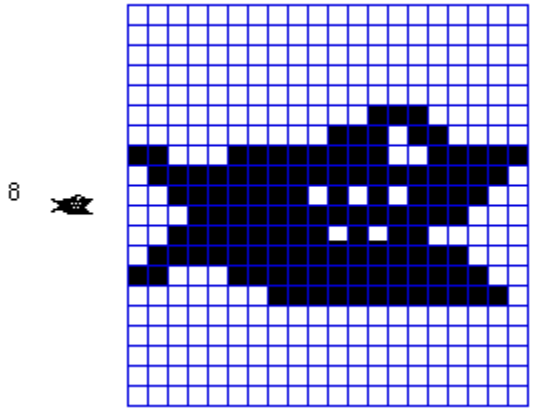


Figure 8: Shape #8 - baby carnivore

(Top: Teacher simulation - Bottom: Student simulation)



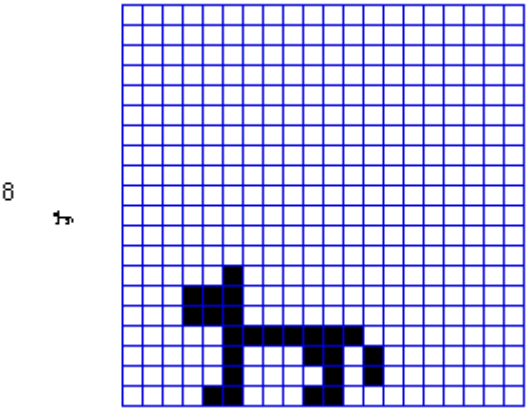


Figure 10: Herbivore and Carnivore Feedings

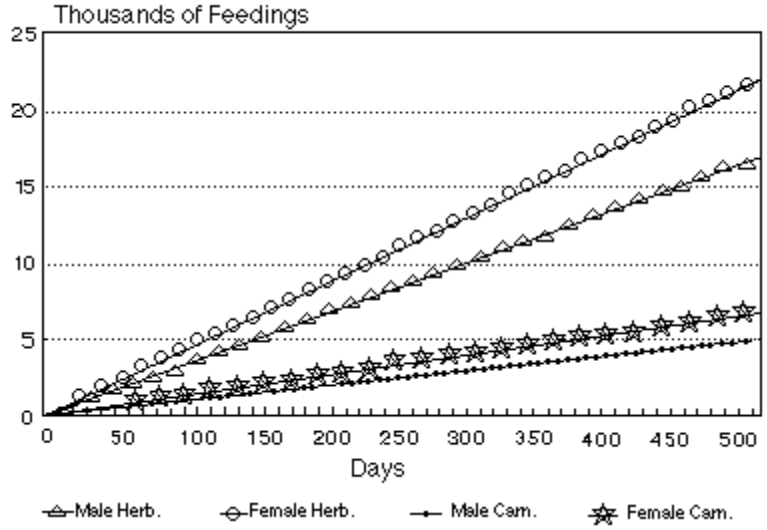


Figure 11: Herbivore and Carnivore Births

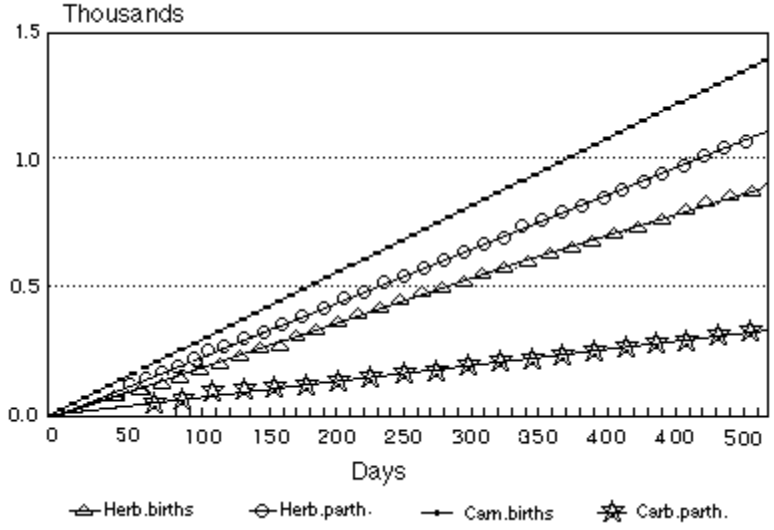


Figure 12: Herbivore Feedings after 112 daily runs

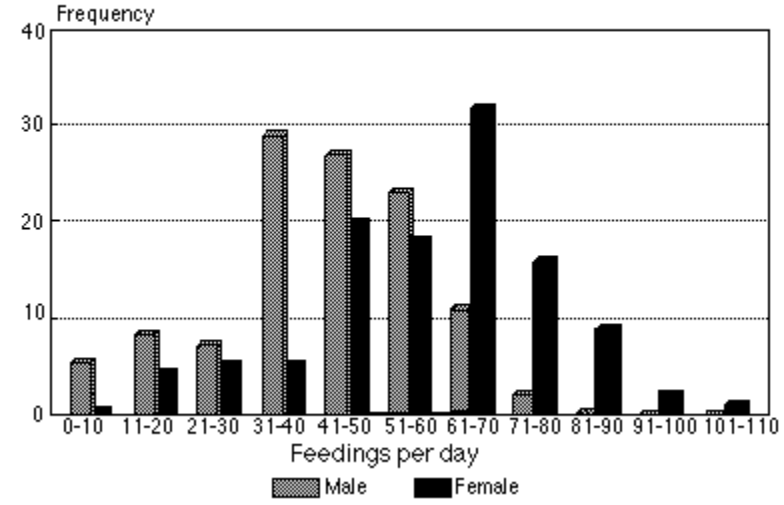


Figure 13: Carnivore Feedings after 112 daily runs