Pasture-ization 2.0

New Mexico

Supercomputing Challenge

Final Report

March 23, 2015

Team Number: 69

School Name: Melrose High School

Area of Science: Biology

Computer Language: Netlogo 5.0.4

Team Member: Ethan Wright

Teacher: Alan Daugherty

Table of Contents

Introduction

As I have come from a farm and ranch community, I have ties to this way of life that is as much a part of me as anything else. I have a passion for this lifestyle, and I wish to help better it for future generations as my ancestors did for me. I think this line of the FFA Creed (Future Farmers of America) by E. M. Tiffany sums it up best.

"*I believe in the future of agriculture, with a faith born not of words but of deeds – achievements won by the present and past generations of agriculturalists; in the promise of better days through better ways . . .*"

In my first year of the challenge, my team and I visited this concept briefly in the programming language Starlogo TNG. Unfortunately we were not able to complete this project because we were all first year participants, just beginning to learn coding skills. Also, we found that we were severely limited as to what we could accomplish in Starlogo TNG.

As I began to decide what my project would be this year, I realized that I wanted to revisit this concept. I decided to make myself finish something that I had started years ago, and to find out if stocking rates could really be calculated. I wished to do a project involving something I was passionate about, and through this project I found that something.

Executive Summary

The purpose of this model is to calculate the number of cattle that can be grazed on a pasture without damaging land, or affecting cattle growth. Currently the most common practice to achieve this goal is to look at a pasture and guess. If it looks good, leave the cattle on, if it doesn't, pull them off. A cattleman will think it looks good if there are plentiful amounts of grass present. There are many problems with this method. One being that it is subjective, there is not a standard of measurement. Another is that if a rancher pulls his cattle off a pasture too early, he is not getting the most out of his land. Most importantly if the cattle are pulled off too late, then overgrazing can occur, damaging pasture land and affecting profits.

I have achieved this goal by creating a Netlogo model. My model incorporates the basic principles of common practices by simulating a pasture and taking into account many different variables. This model shows how stocking rates can be calculated using basic land practice principles. I confirmed my hypothesis with this model. I found that using basic, easily obtainable variable values, stocking rates can be calculated fairly accurate.

Problem Statement

In this project I have been modeling how to fix the problem of overgrazing. Developing a better method for solving this problem will greatly help ranchers maximize profit and achieve optimum land management practices. This is a project that I myself have direct ties to, as many members of my family are farmers and ranchers. Having direct ties to the land, I have directly seen the affect that bad land practices can have. With this project I hope to improve common ranching practices, both within my family and others in the industry.

Method

I have created a model in Net Logo 5.0.4 to simulate grazing conditions. I have put much time and effort into making this model as realistic as possible. This will allow me to more accurately test my hypothesis. In this model, we have simulated a pasture that is a quarter section, a section being 1 square mile, in size. A quarter section is a fairly common pasture size in this area, and the values produced by this pasture size are easy to apply to larger, or smaller pastures. I have coded in variables such as rain, the amount of cattle, how much grass was on the pasture at the starting time, and the rate at which the grass grows. Also, we have coded in a variable with a chooser option (the actual section of the interface in Net Logo that controls this code is called a chooser) that allows the user to tell the model whether the start date is in the winter or during the growing season. I have also coded in monitors in my interface which allow the user to see the values of certain variables.

The variables function as follows: the rainfall variable allows the user to input values specific to their area. This, in turn, factors in with other variables to control the rate at which the grass in the pasture grows. The amount of cattle variable only affects the setup procedure, as that is the code that controls the amount of cattle that are in the pasture. I have another variable that works hand in hand with the rainfall variable, called grass growth percentage, that controls how fast the grass replenishes.

Verification and Validation

I have worked on verification and validation in two different ways. One being that I have been cross-referencing a program called The Grazing Manager, or TGM. This program was developed in Nebraska and is currently being tested in Texas. I will provide a link to this program in the resource section of this report, however, if you wish to explore its features, a software download will be required.

I have also used some of the data and run simulations for our ranch, formally known as Wright Farms and Cattle. Through using the data I have produced on our own ranch, I have been able to see first-hand that this is a valid, and feasible concept.

One problem we had while working through the verification and validation process was that in a real-life application different types of grass grew at different rates in different soil types. To fix this problem, I would have to code in the variable for soil, and essentially give each patch its own characteristics. For my purposes, and to achieve my desired results, I decided to abstract out this variable.

Results

I have obtained results supporting my hypothesis. I have shown that in most real-life applications that stocking rates can be calculated. Most of the data I have collected leans toward the conclusion that grazing management can in fact be scientifically calculated. That is not to say that I have not had some negative results within this model. For instance, this model has no way of planning for natural disasters, such as heavy snows, early or late winters, or sickness.

Every calculation is different, and every real-life application would be case sensitive. Achieving the desired, accurate, results depends on the user doing various calculations throughout the chosen time period.

Conclusions

Throughout my experimentations and calculations I have come to the conclusion that calculating grazing rates is possible. Even though it is possible, however, it may not be practical with one single program. A user may have to run several different models for different situations. Case in point, in some areas soil and forage type may change in parts of one pasture. The terrain could also change.

Obtaining accurate results for one specific pasture would require much work on the part of the user. This is not a one-time calculation and done. This model, if used by a rancher, will require re-calculations and data entry each month, possibly more frequently than that depending on the rate of variable change in a real-life application.

Achievements

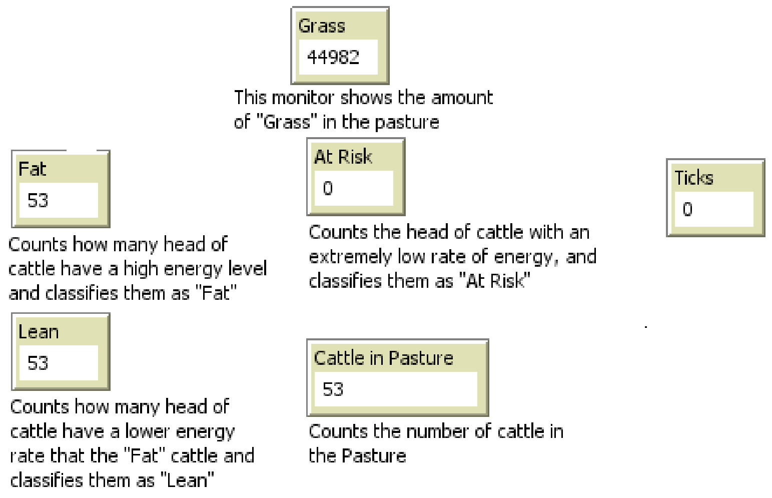
The most significant achievement that has come from this project is the critical thinking and coding skills that I have gained. Every time I sat down to write some code and got a bug, I had to think, and rethink the logic I was using. Sometimes, I had to scrap everything I had just done and look at the problem at hand in a whole new prospective, completely different from how I normally would have gone about it.

Also, I have learned the value that a good team can bring to the table. It has been a challenge this year to make all the deadlines on my own.

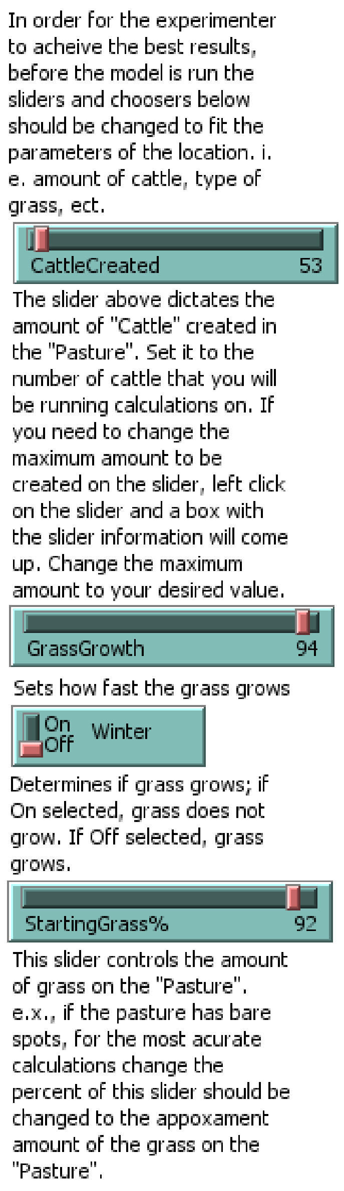
Screenshots

Below I have provided current screenshots of my model interface and code. It may benefit those reading this report to be provided a color copy of the screenshots. By the time competition rolls around I will have fine-tuned some areas of my code, and will provide an updated copy, in color, for the judges at that time.

Interface Screenshots


Grass
44982
This monitor shows the amount of "Grass" in the pasture
Fat
53
Counts how many head of cattle have a high energy level and classifies them as "Fat"
At Risk
0
Counts the head of cattle with an extremely low rate of energy, and classifies them as "At Risk"
Ticks
0
Lean
53
Counts how many head of cattle have a lower energy rate that the "Fat" cattle and classifies them as "Lean"
Cattle in Pasture
53
Counts the number of cattle in the Pasture

This picture is of the monitors in the interface of my model.


In order for the experimenter to acheive the best results, before the model is run the sliders and choosers below should be changed to fit the parameters of the location. i. e. amount of cattle, type of grass, ect.

CattleCreated 53

The slider above dictates the amount of "Cattle" created in the "Pasture". Set it to the number of cattle that you will be running calculations on. If you need to change the maximum amount to be created on the slider, left click on the slider and a box with the slider information will come up. Change the maximum amount to your desired value.

GrassGrowth 94

Sets how fast the grass grows

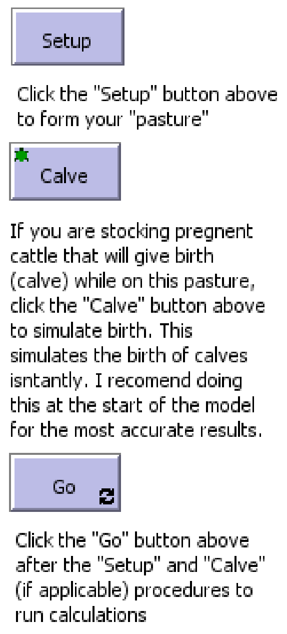
On Off Winter

Determines if grass grows; if On selected, grass does not grow. If Off selected, grass grows.

StartingGrass% 92

This slider controls the amount of grass on the "Pasture". e.x., if the pasture has bare spots, for the most acurate calculations change the percent of this slider should be changed to the appoxament amount of the grass on the "Pasture".

This is a picture of the variable sliders in the interface screen of this model.

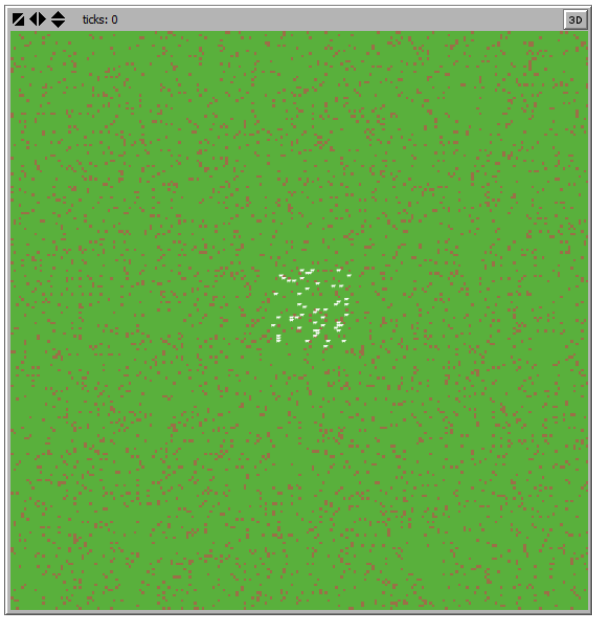


Click the "Setup" button above to form your "pasture"

If you are stocking pregnent cattle that will give birth (calve) while on this pasture, click the "Calve" button above to simulate birth. This simulates the birth of calves isntantly. I recomend doing this at the start of the model for the most accurate results.

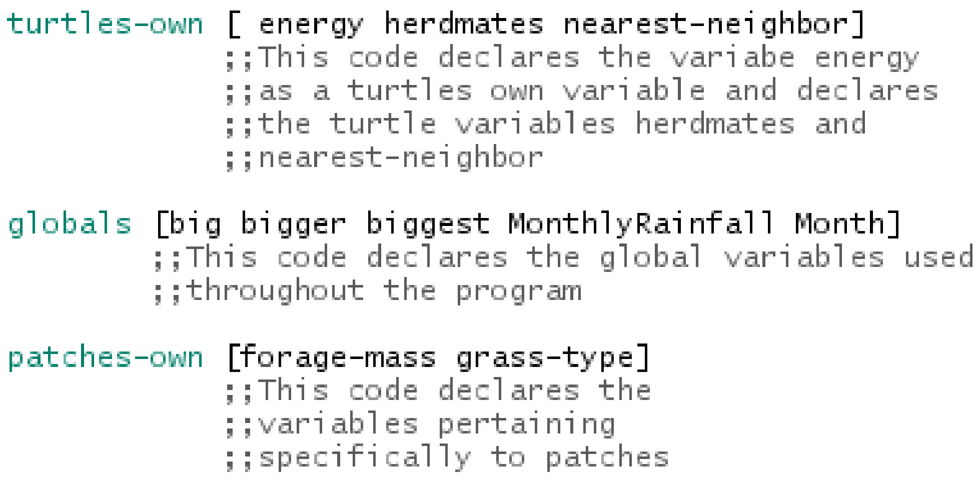
Click the "Go" button above after the "Setup" and "Calve" (if applicable) procedures to run calculations

This is a picture of the procedure buttons in the interface of this model.



This is a picture of the Pasture in the interface of this model. The Green patches represent grass, the brown patches represent areas where there is no grass. The cattle are represented by agents with a white color and shape of cattle.

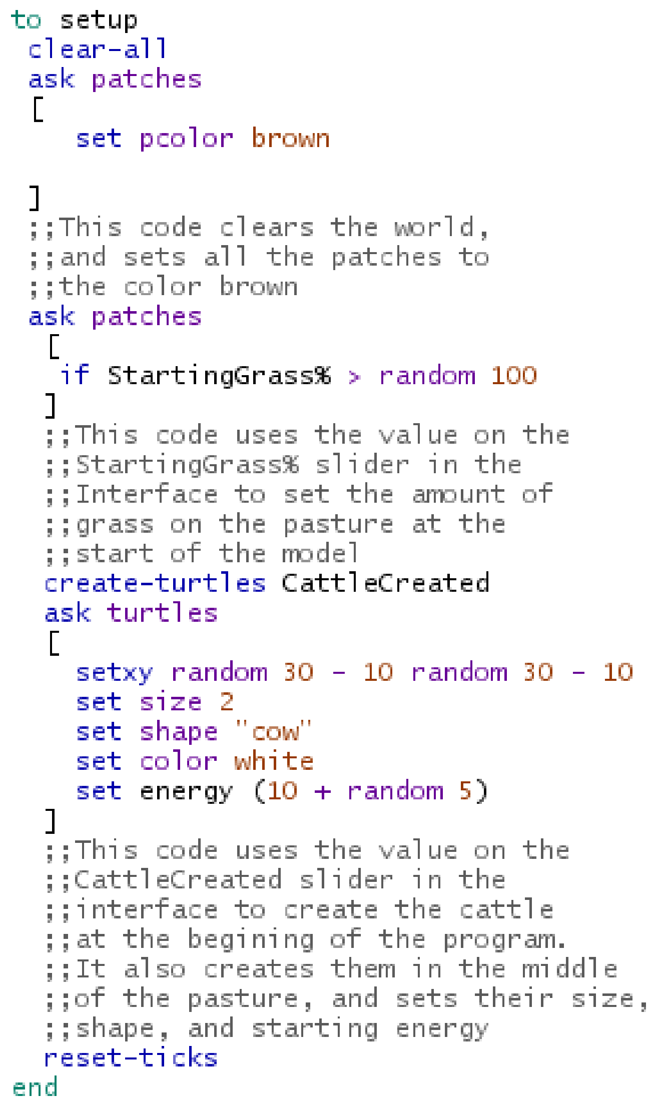
Code Screenshots



```
turtles-own [ energy herdmates nearest-neighbor]
            ;;This code declares the variabe energy
            ;;as a turtles own variable and declares
            ;;the turtle variables herdmates and
            ;;nearest-neighbor

globals [big bigger biggest MonthlyRainfall Month]
        ;;This code declares the global variables used
        ;;throughout the program

patches-own [forage-mass grass-type]
            ;;This code declares the
            ;;variables pertaining
            ;;specifically to patches
```



```
to setup
 clear-all
 ask patches
 [
    set pcolor brown

 ]
 ;;This code clears the world,
 ;;and sets all the patches to
 ;;the color brown
 ask patches
  [
   if StartingGrass% > random 100
  ]
  ;;This code uses the value on the
  ;;StartingGrass% slider in the
  ;;Interface to set the amount of
  ;;grass on the pasture at the
  ;;start of the model
  create-turtles CattleCreated
  ask turtles
  [
    setxy random 30 - 10 random 30 - 10
    set size 2
    set shape "cow"
    set color white
    set energy (10 + random 5)
  ]
  ;;This code uses the value on the
  ;;CattleCreated slider in the
  ;;interface to create the cattle
  ;;at the begining of the program.
  ;;It also creates them in the middle
  ;;of the pasture, and sets their size,
  ;;shape, and starting energy
  reset-ticks
end
```

```
to go
  ask turtles
  [
    herd
    move
    ;; the following line is used to make the turtles
    ;; animate more smoothly.
    repeat 5 [  fd 0.2 ] display
    ;; for greater efficiency, at the expense of smooth
    ;; animation, substitute the following line instead:
    ;; ask turtles [ fd 1 ]
    graze
    setsize
    grow?
  ]
  ;;This code calls the go procedure which
  ;;is made up of several other sub-procedures
  tick

end


to calve
  if energy > 1
   [
     hatch 1
     [
     set size (energy / 4)
     ]
   ]
  if energy < 1
   [
     set color red
   ]
   ;;This code simulates birth of calves
   ;;if pasture is stocked with pregnant
   ;;cattle
end

to move
   fence
   rt random 30 lt random 30
   forward 1
   set energy energy - 1
   ;;This code allows the cattle to move
   ;;and sets the rate of energy loss
   ;;per step
end
```

```
to graze
    let colorofpatchahead  green
    set colorofpatchahead pcolor
    if (colorofpatchahead = brown)
    [
      forward 1
    ]
      if (colorofpatchahead = green)
    [
      set energy energy + 10
      set pcolor brown
      forward 1
      fence
      forward 1
      fence
    ]
    ;;This code tells the cattle what to
    ;;do based on the patch color
end
to setsize
    if energy > 10000
    [
      set size 3
      set color white
    ]
    if energy < 1
    [
      set size 2
      set color white
    ]
    if energy < -1
    [
      set size 1
      set color red
    ]
    ;;This code sets the turtles size,
    ;;which can be seen on monitors in
    ;;the interface
end

to grow?
         ifelse Winter
         [nogrow]
         [grow]
         ;;This coding allows the user
         ;;in the interface to tell the
         ;;model if the start date is in
         ;;the winter season
end
```

```
to grow
   if random 100 < (GrassGrowth * .1)
     [set pcolor green]
   if Month = 10
   [
     if Winter
     [nogrow]
   ]
   ;;This procedure sets the code
   ;;allowing grass to grow during
   ;;the growing season
end


to nogrow
  if random 100 < (GrassGrowth * 0)
    [set pcolor green]
  if Month = 3
  [
    if Winter
    [grow]
  ]
  ;;This procedure stops grass growth
  ;;during the winter months
end
```

```
to SetMonth
  let Year 0
  set Year 0
  if ticks < 721
  [set Month 1 ]
  ;;January
  if ticks > 720 or ticks < 1441
  [set Month 2]
  ;;February
  if ticks > 1440 or ticks < 2161
  [set Month 3]
  ;;March
  if ticks > 2160 or ticks < 2881
  [set Month 4]
  ;;April
  if ticks > 2880 or ticks < 3601
  [set Month 5]
  ;;May
  if ticks > 3600 or ticks < 4321
  [set Month 6]
  ;;June
  if ticks > 4320 or ticks < 5041
  [set Month 7]
  ;;July
  if ticks > 5040 or ticks < 5761
  [set Month 8]
  ;;August
  if ticks > 5760 or ticks < 6481
  [set Month 9]
  ;;September
  if ticks > 6480 or ticks < 7201
  [set Month 10]
  ;;October
  if ticks > 7200 or ticks < 7921
  [set Month 11]
  ;;November
  if ticks > 7921 or ticks < 8641
  [set Month 12]
  ;;December
  if ticks = 8641
  [
    reset-ticks
    set Year (Year + 1)
  ]
  ;;This code uses the tick values
  ;;to determine what month of the
  ;;year that this model is in
end
```

```
to RainFall
  ;let MonthlyRainfall = 0
  ;let Month = 0
  if Month = 1  ;;January
  [set MonthlyRainfall 0.45]
  if Month = 2  ;;February
  [set MonthlyRainfall 0.47]
  if Month = 3  ;;March
  [set MonthlyRainfall 0.71]
  if Month = 4  ;;April
  [set MonthlyRainfall 1.01]
  if Month = 5  ;;May
  [set MonthlyRainfall 2.01]
  if Month = 6  ;;June
  [set MonthlyRainfall 2.58]
  if Month = 7  ;;July
  [set MonthlyRainfall 2.60]
  if Month = 8  ;;August
  [set MonthlyRainfall 2.46]
  if Month = 9  ;;September
  [set MonthlyRainfall 2.09]
  if Month = 10 ;;October
  [set MonthlyRainfall 1.71]
  if Month = 11 ;;November
  [set MonthlyRainfall 0.60]
  if Month = 12 ;;December
  [set MonthlyRainfall 0.61]
  ;;This code sets the MonthlyRainfall
  ;;value based on the average rainfall
  ;;in my area for each month
end
```

```
to herd  ;; turtle procedure
  find-herdmates
  if any? herdmates
    [ find-nearest-neighbor
      ifelse distance nearest-neighbor < 2
        [ separate ]
        [ align
          cohere ] ]
end
```

```
;;; HERDING
to find-herdmates  ;; turtle procedure
  set herdmates other turtles in-radius 6
end
```

```
to find-nearest-neighbor ;; turtle procedure
  set nearest-neighbor min-one-of herdmates [distance myself]
end
```

```
;;; SEPARATE

to separate  ;; turtle procedure
  turn-away ([heading] of nearest-neighbor) 1
end

;;; ALIGN

to align  ;; turtle procedure
  turn-towards average-herdmate-heading 15
end

to-report average-herdmate-heading  ;; turtle procedure
  ;; We can't just average the heading variables here.
  ;; For example, the average of 1 and 359 should be 0,
  ;; not 180.  So we have to use trigonometry.
  let x-component sum [dx] of herdmates
  let y-component sum [dy] of herdmates
  ifelse x-component = 0 and y-component = 0
    [ report heading ]
    [ report atan x-component y-component ]
end

;;; COHERE

to cohere  ;; turtle procedure
  turn-towards average-heading-towards-herdmates 15
end

to-report average-heading-towards-herdmates  ;; turtle procedure
  ;; "towards myself" gives us the heading from the other turtle
  ;; to me, but we want the heading from me to the other turtle,
  ;; so we add 180
  let x-component mean [sin (towards myself + 180)] of herdmates
  let y-component mean [cos (towards myself + 180)] of herdmates
  ifelse x-component = 0 and y-component = 0
    [ report heading ]
    [ report atan x-component y-component ]
end

;;; HELPER PROCEDURES

to turn-towards [new-heading max-turn]  ;; turtle procedure
  turn-at-most (subtract-headings new-heading heading) max-turn
end

to turn-away [new-heading max-turn]  ;; turtle procedure
  turn-at-most (subtract-headings heading new-heading) max-turn
end
```

```
;; turn right by "turn" degrees (or left if "turn" is negative),
;; but never turn more than "max-turn" degrees
to turn-at-most [turn max-turn]  ;; turtle procedure
  ifelse abs turn > max-turn
    [ ifelse turn > 0
        [ rt max-turn ]
        [ lt max-turn ] ]
    [ rt turn ]
end

; set random location
to randomize
  setxy random-xcor random-ycor
  if pcolor = blue        ; if it's on the fence...
    [ randomize ]         ; ...try again
    ;;This code block sets the cattle to
    ;;random positions, if needed
end


to fence  ;; turtle procedure
  ; check: hitting left or right fence?
  if abs [pxcor] of patch-ahead 1 >= 107
    ; if so, reflect heading around x axis
    [ set heading (- heading) ]

     ;[ set heading (- heading) ]
  ; check: hitting top or bottom wall?
  if abs [pycor] of patch-ahead 1 >= 107
    ; if so, reflect heading around y axis
    [ set heading (180 - heading) ]

     if abs xcor > 106 [set heading towardsxy 0 0]
     if abs ycor > 106 [set heading towardsxy 0 0]

end
```

References

http://www.wrcc.dri.edu/cgi-bin/cliMAIN.pl?nm1939

http://www.agreninc.com/projects.php?proj=5

https://extension.usu.edu/files/publications/publication/NR_RM_04.pdf

http://www.ag.ndsu.edu/archive/streeter/2006report/aums/Doing%20the%20Math.htm

http://www.okrangelandswest.okstate.edu/files/grazing%20management%20pdfs/F-2871web.pdf

Doranne Publishing Farm Management Guide

Gillespe Modern livestock and Production