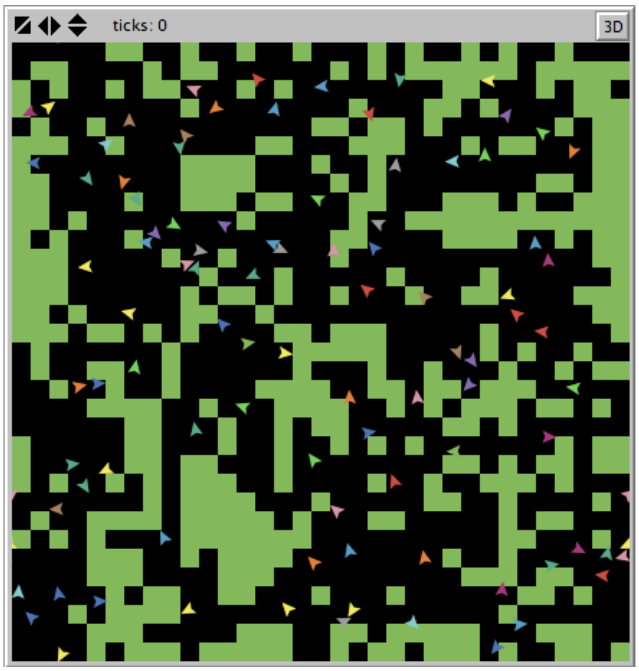
Agents That Move

NetLogo Guide 2

Turtles - Breeds



John Hayward
Church Growth Modelling
www.churchmodel.org.uk

Church Growth Modelling

Turtles

An **agent** is an individual entity, or being, that can follow instructions and modify its behaviour in response to other agents and the environment.

Unlike a patch (or cell) a **turtle** is an agent that can move around the world. Thus it can interact with the patches near where it is located and it can interact with other agents in their vicinity.

Thus the restriction that patches have of only having 8 (or 4) fixed neighbours is now restricted. Neighbours and relationships can change over time.

Like patches turtles have properties.

You are not restricted to the name turtle you can breed your own species!

1 Basics

1.1 Turtles

A turtle is an agent that moves. Like a patch it has coordinates. But unlike a patch its coordinates are real numbers not an integer. Thus they do not just move from patch to path on the world but can be located at a particular position in a patch.

This leads to a difference in what is meant by neighbour. For a patch it is the 8 surrounding patches. For a turtle it is the radius of influence. That is its neighbours are the turtles that are within a certain distance which the user can define.

The other major difference is that you can define more than one type of turtle. For this reasons turtles can be given their own names to distinguish them

Other than that a turtle is much like a patch. It can have properties or variables that define what it is. These properties can change as time progresses.

1.2 Turtles Moving

We will have a simple model of turtles moving about the world. As before you will need a *set up* procedure and a *go* procedure. All this model will do is set up 10 turtles and move each one unit at a time in a random direction.

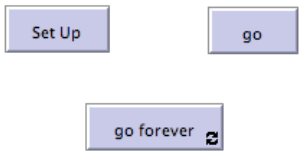
Start a new file and type:

```
; the set up
to setup
  clear-all
  setup-turtles
end

to setup-turtles
  create-turtles 10
  ask turtles
  [
    setxy random-xcor random-ycor
  ]
end

; main procedure
to go
  move-turtles
end

to move-turtles
  ask turtles
  [
    right random 360
    forward 1
  ]
end
```

The italic words are user defined, the rest are part of the language. The semicolon is the start of a comment.

Initially there are 10 turtles spread randomly over the world (each is asked to execute setxy random-xcor random-ycor). Then on each move the turtle turns to an angle anywhere between 0 and 360, then asked to move one space forward.

On the interface place buttons, for setup, go and go forever (using go and the forever boxed checked):



Run the model. I.e. click Set Up then click go forever.

If all works correctly you will see the 10 turtles move around the screen randomly and very fast. You may need to slow them down to see their movements!

2 Rabbits Eating Grass

2.1 Breed

Time to think of a specific model. Let our turtles be rabbits and let the rabbits eat grass to survive. It makes sense to call the agent a rabbit rather than a turtle. This is done with the *breed* command. It must go at the top of the program. Type:

```
breed [Rabbits Rabbit]
```

You have now defined a moving agent called a rabbit. The reason for the plural and the singular as that both are used in turtle commands. Sometimes the command is on all agents sometimes just on one.

Now in the rest of your program everywhere you see the word “turtles” replace it with the word “Rabbits”. Use find and replace to do it quickly. Your program should now look like:

```
breed [Rabbits Rabbit]

; set up
to setup
  clear-all
  setup-Rabbits
end

to setup-Rabbits
  create-Rabbits 10
  ask Rabbits
  [
    setxy random-xcor random-ycor
  ]
end

 ; main procedure
to go
  move-Rabbits
end

to move-Rabbits
  ask Rabbits
  [
    right random 360
    forward 1
  ]
end
```

One of the powerful things about NetLogo is that the commands for turtles can all be recycled for your new agent. Thus create-Rabbit is the NetLogo command create-turtle with your name substituted.

Of course if you run it, it will be identical to before, as the model has not changed.

2.2 Grass as Patches

A patch will either contain grass or not. So its data model is simple (and identical to Conway's Game of Life). Place just after the breed command:

```
patches-own
[
  has-grass?
  ; true if grass false if no grass
]
```

We will use green for the grass. Place somewhere near the setup command a procedure to set up the patches:

```
to setup-patches
  ask patches
  [
    set has-grass? true
    set pcolor green
  ]
end
```

Add setup-patches to the main set up procedure (bold below)

```
to setup
  clear-all
  setup-Rabbits
  setup-patches
end
```

Running the program has the rabbits running over the graph but little else.

2.3 Interaction of Agent and Patch

The main interaction is that rabbits eat grass! As a result the rabbits gain energy. Thus a model of a rabbit is now needed. We will let a rabbit have energy. Add a data model for Rabbit after the breed command:

```
Rabbits-own
[
  energy
]
```

When they eat grass they gain energy, but the grass is gone. Make this a procedure. Add:

```
to eat-grass
   ask Rabbits
   [
     if has-grass?
     [
       ; grass is eaten
       set pcolor  black
       set has-grass? false
       ; rabbit gains energy
       set energy (energy + 10)
     ]
   ]
end
```

The rabbit gains 10 units of energy, the grass disappears and the patch is coloured black. Why 10? This is should be a parameter of the model, so different interpretations of energy can be explored.

Note that for each rabbit it interprets if *has-grass?* for the patch the rabbit is on. So no messy programming with coordinates is needed.

Place eat-grass in the main procedure:

```
to go
  move-Rabbits
  eat-grass
end
```

Run the simulation to see the grass being eaten. The grass is quickly eaten and the rabbits run over bare earth! They should die of starvation. Next we need to update the move rabbit procedure for them to lose energy as they move. One simple command to reduce energy with movement

```
to move-Rabbits
  ask Rabbits
  [
    right random 360
    forward 1
    set energy (energy - 1)
  ]
end
```

Thus a rabbit uses one unit of energy for each unit of distance it moves. All that will happen now when the grass runs out is that it gets negative energy levels. Once energy drops to zero the rabbit should die. Turtles unlike patches can be removed from the world, using the *die* command. Type in the following:

```
to check-death
  ask Rabbits
  [
    if energy <= 0 [die]
  ]
end
```

Then call this procedure in the main procedure

```
to go
  move-Rabbits
  eat-grass
  check-death
end
```

Running the model you will see the rabbits quickly die out after the grass is eaten. Often there is some grass left which the remaining rabbits have not had the energy to get to.

2.4 Renewing the Grass

For a sustainable solution the grass needs to re-grow. The way to control the rate of growth is to use probability. We will say at any given time there is a 3% chance of grass re-growing on a patch. Type

```
to regrow-grass
  ask patches
  [
    if random 100 < 3
    [ set pcolor green
      set has-grass? true
    ]
  ]
end
```

The command is purely on the patches. A random number from 0..99 is selected, only if it is 0, 1, 2 does the grass re-grow. Thus a 3% chance of re-growing

Call the procedure in the main one

```
to go
  move-Rabbits
  eat-grass
  check-death
  regrow-grass
end
```

You will get more interesting results if you generate a 100 rabbits initially. Change setup rabbits to do this.

Run the model. You should find now that neither the grass nor the rabbits die out, and a sustainable situation results.

2.5 Reproducing Rabbits

The final piece of code is to allow rabbits to reproduce. Here the model is if they have more than 50 units of energy they will produce one new rabbit. In the process they will give that rabbit 50 units of energy, but lose 50 units themselves. **Add** to the model:

```
to reproduce
  ask Rabbits
  [
    if energy > 50
    [
      set energy energy - 50
      hatch 1 [ set energy 50 ]
    ]
  ]
end
```

I imagine you can see a few flaws in this model of reproduction!

The key word hatch is the NetLogo command for producing a new turtle. In this case it is a rabbit because a turtle always reproduces its own kind.

Call this procedure in the main one:

```
to go
  move-Rabbits
  eat-grass
  reproduce
  check-death
  regrow-grass
end
```

Now re-run the model. Again the solution is sustainable but it looks if there is less grass and more rabbits.

The full code to date appears overleaf. You may wish to check you have everything correct. Ignore the fact that some bits are in bold. They are used in section 3.

```
breed [Rabbits Rabbit]

Rabbits-own
[
  energy
]


patches-own
[
  has-grass?
  ; true if grass false if no grass
]

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; set up
to setup
  clear-all
  setup-Rabbits
  setup-patches
end

to setup-patches
  ask patches
  [
    set has-grass? true
    set pcolor green
  ]
end

to setup-Rabbits
  create-Rabbits 100
  ask Rabbits
  [
    setxy random-xcor random-ycor
  ]
end

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

 ; main procedure
to go
  move-Rabbits
  eat-grass
  reproduce
  check-death
  regrow-grass
end

; Rabbits move one space at a time is a random
; direction and lose energy in the process
to move-Rabbits
  ask Rabbits
  [
    right random 360
    forward 1
    set energy (energy - 1)
  ]
end

; Rabbits eat grass, gain energy but the grass on the
patch is lost
to eat-grass
   ask Rabbits
   [
     if has-grass?
     [
       set pcolor  black
       set has-grass? false
       set energy (energy + 10)
     ]
   ]
end

; If rabbits have no energy left they die
to check-death
  ask Rabbits
  [
    if energy <= 0 [die]
  ]
end

; this is the re-growth rate of the grass
to regrow-grass
  ask patches
  [
    if random 100 < 3
    [ set pcolor green
      set has-grass? true
    ]
  ]
end

; Rabbits reproduce if they have enough energy
; they produce one new rabbit, but lose energy
; in the process
to reproduce
  ask Rabbits
  [
    if energy > 50
    [
      set energy energy - 50
      hatch 1 [ set energy 50 ]
    ]
  ]
end

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

3 Experiments

3.1 Parameters

To carry out experiments on a model it is helpful to generalise any elements that up to now have their numbers fixed. For example you are generating 100 rabbits when setting up. This should be set by the user and thus be made a **parameter** of the model. All the potential parameters of the model are in bold on the previous page.

setup-Rabbits should be changed to:

```
to setup-Rabbits
  create-Rabbits initial-rabbits
  ask Rabbits
  [
    setxy random-xcor random-ycor
  ]
end
```

The model will no longer check as initial-rabbits is not yet defined.

Do the same with the grow rate of the grass

```
to regrow-grass
  ask patches
  [
    if random 100 < chance-of-regrowth
    [ set pcolor green
      set has-grass? true
    ]
  ]
end
```

chance-of-regrowth is the probability (as a percentage) that at any time any empty patch becomes filled with grass again. (It ignores whether there are neighbouring ones to spread the grass – another flaw in the model)

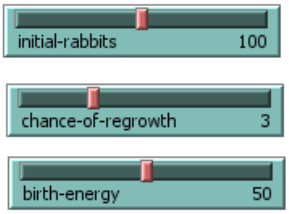
Do the same with the ability of rabbits to reproduce:

```
to reproduce
  ask Rabbits
  [
    if energy > birth-energy
    [
      set energy energy -  birth-energy
      hatch 1 [ set energy  birth-energy ]
    ]
  ]
end
```

The loss of energy and the energy for the new rabbit need not be the same as the critical point. There are 3 possible parameters here. However we are keeping things simple.

Although there are possible parameters with the way a rabbit moves, the distance, direction and the energy lost, the energy in the grass. However we will not do these for the sake of time.

To define the 3 new parameters place sliders on the interface.



initial-rabbits is on a scale 0..200 with a normal value of 100.

chance-of-regrowth is on a scale 0..10 with a normal value of 3

birth-energy is on a scale 0..1 with a normal value of 50.

Thus model is as before. Run to check it still works

3.2 Global Variables

The sliders define parameters that are global constants. That is they are values that affect the whole model, all agents.

It is possible to have global variables, variables that change with the running of the model that can in turn affect agents. There are no obvious ones in this model. But to perform experiments it will be helpful to have some running totals of rabbits, grass etc. These are also global variables.

Just after the breed command place the following global variables:

```
globals
[
  total-energy
  total-Rabbits
  average-energy
  total-grass
 total-bare
  fraction-grass
]
```

Thus we intend to have the total number of rabbits, fraction of grass and the average energy of a rabbit. The total amount of grass, bare patches and the total energy of all rabbits will be used to help construct these.

Global variables like agents need to be updated. The update procedure can be used in the setup as well as in the main procedure. Firstly one for the rabbit globals:

Add in (anywhere in the procedures section)

```
to update-rabbit-globals
  set total-Rabbits (count Rabbits with [energy > 0])
  set total-energy (sum [energy] of Rabbits)
  ifelse total-Rabbits > 0
  [ set average-energy (total-energy / total-Rabbits)]
  [ set average-energy (0)]
end
```

- The first command counts all live rabbits and places it in total
- The second sums up the energy of all rabbits. Notice that NetLogo does not require you to do any loops etc for this!
- The third command divides them to get the average energy of a rabbit. The only reason for the if statement is to avoid a divide by zero if all the rabbits die out!

To deduce the fraction of grass, count the grass patches, count the bare patches and divide appropriately.

```
to update-grass-globals
 set total-grass (count patches with [has-grass?])
 set total-bare (count patches with [not has-grass?])
 set fraction-grass (total-grass / (total-grass + total-bare))
end
```

Place both updates on the *setup* and the main *go* procedures.

```
to setup
  clear-all
  setup-Rabbits
  setup-patches
  update-rabbit-globals
  update-grass-globals
end
```

```
to go
  move-Rabbits
  eat-grass
  reproduce
  check-death
  regrow-grass
  update-rabbit-globals
  update-grass-globals
end
```

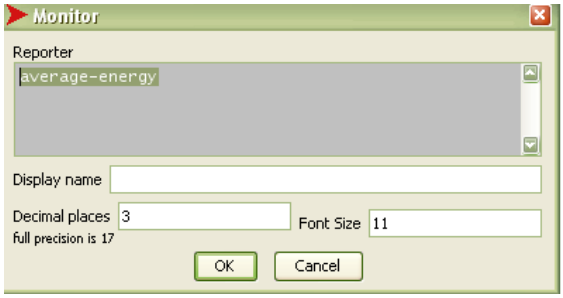
Just run the model to check it still works. Without output there is nothing much to see!

3.3 Output

For output you should have monitors and graphs for average energy, total rabbits and fraction grass.

The full details of how to do these was given in the introduction handout.

The monitors are straightforward; you just report the name of the global variable you require. Once clicking the monitor and "Add" you get the dialogue:



Type in average-energy and cut the number of decimal places down. Do the same for the other three.

For the graphs some extra code is needed to set up the graphs and to update them. Add the procedures at the end of your model:

```
to setup-energy-plot
 set-current-plot "Average Energy"
end

to setup-totals-plot
 set-current-plot "Total Rabbits"
end

to setup-grass-plot
 set-current-plot "Fraction Grass"
end
```

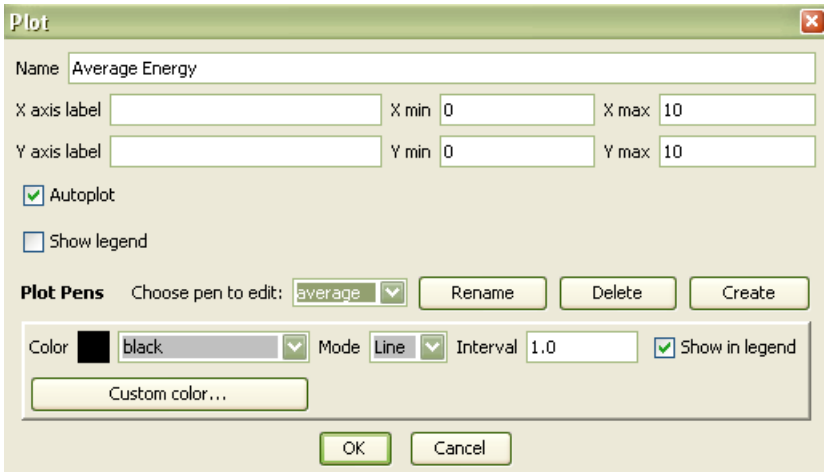
```
to do-energy-plot
 set-current-plot "Average Energy"
 set-current-plot-pen "average"
 plot average-energy
end

to do-total-rabbits-plot
 set-current-plot "Total Rabbits"
 set-current-plot-pen "total rabbits"
 plot total-Rabbits
end

to do-fraction-grass-plot
 set-current-plot "Fraction Grass"
 set-current-plot-pen "fraction grass"
 plot total-Rabbits
end
```

These lead to the *setup* and *go* being updated (add tick as well).

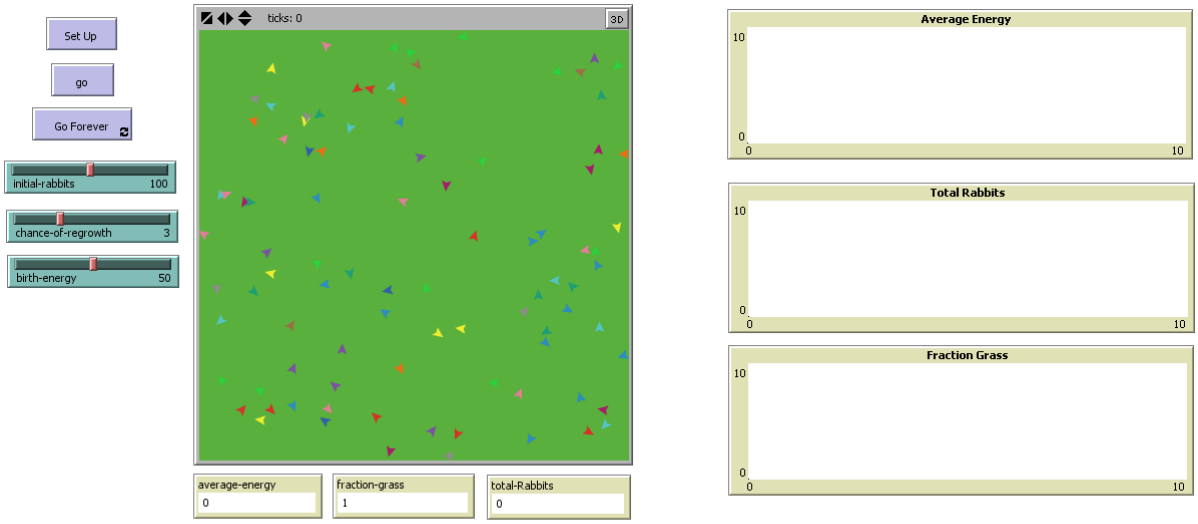
```
to setup
  clear-all
……
  setup-energy-plot
  setup-totals-plot
  setup-grass-plot

  do-energy-plot
  do-total-rabbits-plot
  do-fraction-grass-plot
end
```

```
to go
…..
  regrow-grass
  tick
  update-rabbit-globals
  update-grass-globals

  do-energy-plot
  do-total-rabbits-plot
  do-fraction-grass-plot
end
```

On the interface level three graphs need to be set up. The graph reflects the graph name and the pen reflects the pen name of the code (use create). For example Average Energy is:



The interface should now look like:



And the model should still run.

As a final check you code should look like:

```
breed [Rabbits Rabbit]

globals
[
  total-energy
  total-Rabbits
  average-energy
  total-grass
  total-bare
  fraction-grass
]

Rabbits-own
[
  energy
]


patches-own
[
  has-grass?
  ; true if grass false if no grass
]

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; set up
to setup
  clear-all
  ; set up model
  setup-Rabbits
  setup-patches

  ;update globals
  update-rabbit-globals
  update-grass-globals

  ;setup plots
  setup-energy-plot
  setup-totals-plot
  setup-grass-plot

  ;update plots
  do-energy-plot
  do-total-rabbits-plot
  do-fraction-grass-plot
end

to setup-patches
  ask patches
  [
    set has-grass? true
    set pcolor green
  ]
end



to setup-Rabbits
  create-Rabbits initial-rabbits
  ask Rabbits
  [
    setxy random-xcor random-ycor
  ]
end

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

 ; main procedure
to go
  ;do the model
  move-Rabbits
  eat-grass
  reproduce
  check-death
  regrow-grass
  tick

  ; update globals
  update-rabbit-globals
  update-grass-globals

  ;update plots
  do-energy-plot
  do-total-rabbits-plot
  do-fraction-grass-plot
end

; Rabbits move one space at a time is a random
; direction and lose energy in the process
to move-Rabbits
  ask Rabbits
  [
    right random 360
    forward 1
    set energy (energy - 1)
  ]
end

; Rabbits eat grass, gain energy but the grass on the patch
is lost
to eat-grass
    ask Rabbits
    [
      if has-grass?
      [
        set pcolor  black
        set has-grass? false
        set energy (energy + 10)
      ]
    ]
end

; If rabbits have no energy left they die
to check-death
  ask Rabbits
  [
    if energy <= 0 [die]
  ]
end

; this is the re-growth rate of the grass
to regrow-grass
  ask patches
  [
    if random 100 < chance-of-regrowth
    [ set pcolor green
      set has-grass? true
    ]
  ]
end

; Rabbits reproduce if they have enough energy
; they produce one new rabbit, but lose energy
```

```
; in the process
to reproduce
  ask Rabbits
  [
    if energy > birth-energy
    [
      set energy energy - birth-energy
      hatch 1 [ set energy  birth-energy ]
    ]
  ]
end

; update the globals associated with Rabbits
to update-rabbit-globals
   set total-Rabbits (count Rabbits with [energy > 0])
   set total-energy (sum [energy] of Rabbits)
   ifelse total-Rabbits > 0
   [ set average-energy (total-energy / total-Rabbits)]
   [ set average-energy (0)]
end

; globals for grass
to update-grass-globals
  set total-grass (count patches with [has-grass?])
  set total-bare (count patches with [not has-grass?])
  set fraction-grass (total-grass / (total-grass + total-bare))
end


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


;; plotting

to setup-energy-plot
 set-current-plot "Average Energy"
end

to setup-totals-plot
 set-current-plot "Total Rabbits"
end

to setup-grass-plot
 set-current-plot "Fraction Grass"
end


to do-energy-plot
 set-current-plot "Average Energy"
 set-current-plot-pen "average"
 plot average-energy
end

to do-total-rabbits-plot
 set-current-plot "Total Rabbits"
 set-current-plot-pen "total rabbits"
 plot total-Rabbits
end

to do-fraction-grass-plot
 set-current-plot "Fraction Grass"
 set-current-plot-pen "fraction grass"
 plot total-Rabbits
end
```

3.4 Results

1. Show that the model gives a sustainable balance between the amount of grass and the number of rabbits.
2. Show that increasing the energy at which rabbits give birth increases the average energy of the rabbits but does not affect the general equilibrium values of grass and rabbits. What is the interpretation of this?
3. Show that lowering the birth energy lowers the average energy of the rabbits, and that the *variability* in the numbers of rabbits and grass patches increases.
4. Show that increasing the chance of re-growth of the grass increases the number of rabbits and grass but does not affect the average energy of the rabbits.
5. Show that starting with different numbers of rabbits does not affect the equilibrium value of grass and rabbits.
6. Are there low non-zero values of the parameters where the rabbits could die out near the beginning or die out much later on?
7. What is the effect of very high values of re-growth and birth energy?
8. What is the effect of very high initial numbers of rabbits?

List all the flaws/poor assumptions in this model