StarLogoTNG Biology Curriculum Teacher Guide

Overview

The StarLogoTNG Biology Curriculum is a set of activities that centers on the themes of ecology and evolution. It uses the simple interactions of carrots, rabbits, and wolves, along with various abiotic factors, to highlight the concepts inherent in decentralized systems that are traditionally difficult for students to understand. The projects require students to learn simple programming with the aim of fostering computer usage in science, but the major focus of each activity is system observation and data analysis.

The curriculum is intended as a set of supplementary activities for the normal ecology and evolution units. While it may be used as a stand-alone unit, there may be details regarding each topic that have been removed for modeling purposes. Thus, it is recommended that teachers intersperse traditional teaching days between activities to cover any missing detail, as well as to give students time to digest the material.

Unit Summaries

Lesson 1 begins with just carrots, focusing on the relationships between the carrots and the abiotic elements of the ecosystem. Because the rest of the lessons deal focus primarily on biotic factors, this introductory lesson serves as a reminder that the abiotic factors are just as important. There is no programming involved; instead, the lesson introduces the ideas of modeling and simulation by allowing students to play with sliders and observe the graphs. The main goal of the lesson is to foster the idea of methodical scientific inquiry, whereby students must form a hypothesis and support/ contradict it with collected data. The activity also gives students an easy transition into the StarLogoTNG user interface.

Lesson 2 takes a step back from the biology theme with the VANTS (virtual ants) activity. This activity allows students to create visual designs on the StarLogoTNG SpaceLand using only very few simple commands. On the programming side, VANTS shows that the computer requires specific commands written in a specific way; on the biology side, VANTS highlights the emergent behaviors that arise from each agent performing its own task. With this activity, students are introduced to organization of StarLogo blocks, block canvas, and language.

Lesson 3 begins the ecosystem modeling, requiring the students to program their own rabbits. During this process, students learn the essential procedures that rabbits in the model follow (hop, eat, reproduce, die) that are necessary for a simple functioning ecosystem. While the rabbits are not as complicated as some of the later models, the step-by-step programming tasks give a behind-the-scenes look at the code and teach the students the thought process involved in building a model. After programming, students can observe a classic predator-prey relationship between the rabbits and carrots.

Lesson 4 introduces a gaming aspect that serves the dual purpose of entertaining the students while bringing up the role of humans in the ecosystem. The code now includes a small wolf population, and students program a first person hunter character that shoots the wolves. The model begins with a stable configuration, which students disturb by simply decreasing the age required for wolves to reproduce. The resultant overpopulation tends to overeat and wipe out the rabbits, so the hunters are used to stabilize the population. This lesson brings up ethical issues of hunting and the effects of human interactions with the ecosystem.

Lesson 5 introduces the ideas of competition and selection. For simplicity, the model has returned to being only carrots and rabbits. The activity begins with rabbits having a "color gene" that gives them coats of varying shades of blue. The color gene has no other effect and thus creates no selective advantage. Over time the rabbit population still converges to a single color due to genetic drift, but the color that survives changes between trials. Then students link the color gene first to rabbit speed and then to energy loss, demonstrating directional and stabilizing selection. There is also a "flood" button that speeds up the process by killing 90% of the population, creating a population bottleneck. The goal of this lesson is to emphasize the chance and competition as two very different driving forces behind evolution.

Lesson 6 brings together the driving forces behind evolution with the actual mechanism of mutation. In this project students can change the mutation rate and temperature via sliders. The energy loss in this scenario is now also related to temperature, with exponentially higher energy loss at lower temperatures. The key concept in this lesson is that genetic variation is important as a cushion against changes in the environment, and what is considered "fit" in one set of conditions may be considered "unfit" in another. The code is written such that functionality for more genes can be easily added, so the lesson can be extended to cover other concepts. A possible activity is to allow students to create their own gene and incorporate it into the model using the experience of the previous lessons, but it is important to gauge the level of understanding that they have gained.

Teaching Tips and Notes

Biology Modeling:

- The ecosystem models are *very* temperamental due to the nature of small populations. If you would like to change the model, don't be discouraged if it isn't stable on the first try. Varying parameters makes a *huge* difference on the model's performance.
- It is important to remind the students that the models are not a perfect depiction of real life, so always take the opportunity to discuss, "How is this model a good representation? How is it bad?"
- Tie in complex systems concepts whenever possible and try to bring these concepts back into other units during the course of the year. Diffusion/osmosis, proteins/DNA,

human physiology, etc. are all great examples of emergent behaviors from decentralized systems.
- Think of ways to test the students at the end of the unit. It is important for the students to feel like these exercises are not a waste of time.

StarLogoTNG:
- Students really enjoy being creative when using StarLogoTNG, so it is important to take every opportunity to make the projects their own. While it is difficult with these models to stray too far while getting the same message across, little touches like allowing the students to change terrain or program monsters and cartoons instead of rabbits and wolves already gets them excited. Exploration time also allows students to discover their own tricks and satisfy their curiosities.
- You may want to ask students to work in pairs at a computer, designating roles of "driver" and "navigator" that switch every 15-20 minutes. The driver controls the mouse and keyboard, while the navigator verbally directs the driver's actions using the activity handouts. This will be particularly useful for activities like programming the rabbits, where a support system can help sort out many confusions of programming for the first time. Activities like the hunter may be done in pairs or alone, to give students more individual play-time. However, if you choose to have students working on individual computers, you may still want to designate "buddies" so that students can share projects and ask questions.
- Instruct students to shut off their monitors when you need their undivided attention.
- Encourage students to "Save Next Version" if they create any code, just in case the program freezes unexpectedly.

About the Guide

Each lesson includes the following:
- **Goals:** The overarching theme of the activity.
- **Biology Concepts:** Concepts in the biology curriculum that are covered.
- **StarLogoTNG Programming Concepts:** Programming concepts and elements of StarLogoTNG that are covered.
- **Materials:** Materials needed for the activity.
- **About the Model:** Technical information about the model and code. This description is very detailed and for reference only.
- **Possible Modifications:** Changes that can be easily made to the model to support different goals (of course there are many other possible modifications of varying difficulty to implement).
- **Suggested Teacher Guide:** Outline of a suggested lesson.
- **Student Worksheet (printout):** Student worksheets for distribution.

Project files include the following:
- ***project.sltng***: Starter code for students to do the activity.
- ***project-sol.sltng:*** Solutions code for teacher reference. Contains all completed code in programming portions, as well as detailed commenting of all procedures.

LESSON 1: ECOSYSTEM MODELING

Goals:
The goal of this introductory project is to understand the value of using models and simulations to test hypotheses that may be difficult to test in real life. Students should learn to formulate hypotheses, perform methodical tests, and present data-driven results. In particular, students should realize the importance of sequentially isolating parameters to pinpoint the source of a phenomenon. In terms of understanding ecosystems, students should realize the importance of abiotic factors in ecosystem functioning and how these factors can limit possible growth.

Biology Concepts:
- Population modeling
- Abiotic elements
- Limiting factors
- Carrying capacity
- Exponential vs. logistic growth
- Density dependent regulation
- Density independent regulation

StarLogoTNG Programming Concepts:
- Spaceland exploration (camera views, model speed, terrain editing, agent windows, etc.)
- Setup/ Forever buttons
- Sliders
- Graphs

Materials:
- Starter code: *1-carrots.sltng*
- Student worksheet and graph paper (in case students want to graph behaviors)
- Projector/ computer for demo
- Blackboard/whiteboard/large paper for brainstorming

About the Model:
The model deals with carrots whose growth rates are determined by five changeable (and one preset) parameters. These parameters can be categorized as either density dependent or density independent. Carrot energy production rate and carrot reproduction rate is determined by the density dependent factors, while carrot death rate is determined by the density independent factors.

SETUP:
The model begins with 50 “carrots,” which follow carrot procedures for survival. There is also one “environment” agent that does unseen work of updating nutrient nuggets (explained below) and performing some calculations. The “nutrient nuggets” created by the “environment” are symbolized by lime-colored patches and are only used to keep track of nutrients.

RESET DEFAULT VALUES:
While the model should begin with default slider values, should they be changed, it is simple to restore the default values with the "reset default values" button. Default values are: sunlight = 100, temperature = 65, soil water level = 50, soil nutrient level = 100, soil nutrient distribution = 25.

FOREVER:
Carrot procedures:

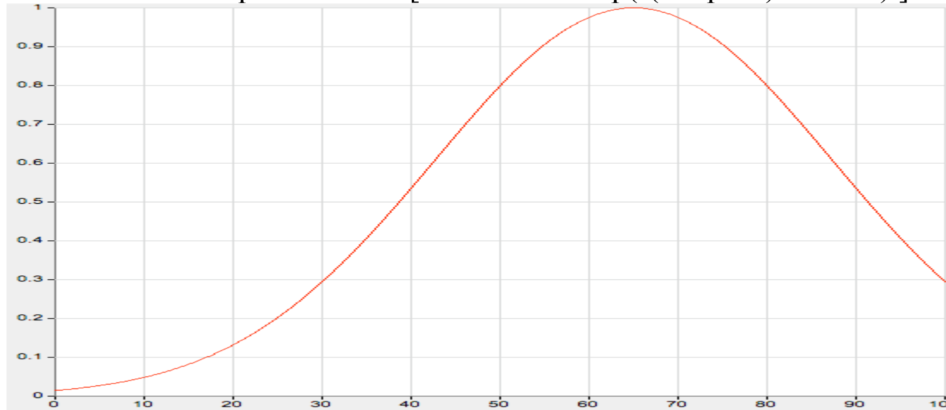
- Increase age +1
- Decrease energy (-0.5)
- Update visual height
- Produce energy +2 (or less depending on density-dependent factors)
- Reproduce (if energy > 10 and fewer than 10 carrots in radius 10)
- Die (if age > 20 or energy < 0, or more depending on density-independent factors)

SLIDERS:
Density dependent factors:

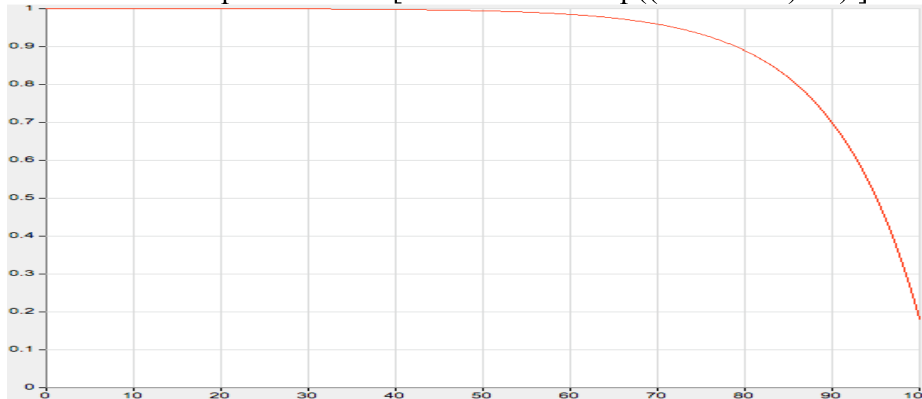
- *Reproduction:* If carrots have at least energy 10, they can reproduce if space allows.
  - Space (preset value): Carrots are limited to 10 carrots within a radius of 10. Outside of this limit, carrots cannot reproduce.
- *Energy production:* Carrots can produce up to maximum energy per iteration of +2. However, low sunlight, soil water, and soil nutrients can limit the energy production to a fraction. Of those three parameters, whichever creates the lowest fraction is the limiting factor, and only that much energy is produced.
    - (Ex. Sunlight is limited to fraction 0.3, soil water is limited at fraction 0.5, and soil nutrient not limited (fraction 1). Therefore sunlight is the limiting factor, and energy production is 0.3*2 = +0.6 instead of +2.)
  - Sunlight (slider): Carrots are limited in growth when there is competition for sunlight within radius of 20.
    - (Ex. If sunlight level is at 100 and there are 20 carrots within radius 20, then [sunlight/ #carrots] = 100/20 = 5. Since 5 > 1, it is not limiting. If sunlight level is at 10, then [sunlight/ #carrots] = 10/20 = 0.5. Since 0.5 < 1, it is limiting, and the sunlight fraction is 0.5.)
  - Soil water level (slider): Carrots are limited in growth when there is competition for water within radius of 20 (same limiting behavior as sunlight).
  - Soil nutrient level and distribution (sliders): Nutrients are distributed according to nutrient "nuggets." The soil nutrient distribution determines how many of these nuggets there are, and the total soil nutrient level is distributed amongst the nuggets (so each nugget has [soil nutrient level/ soil nutrient distribution] amount of available nutrients). During each iteration each nearby carrot that uses that nugget decreases the available nutrients at that nugget by 1 (unless it is used up), and then the available nutrients are replenished.
    - (Ex. If there is a soil nutrient level of 100 and a soil nutrient distribution of 25, then there are 25 nutrient nuggets, each with available nutrient level 4. During each iteration, 4 carrots can use each nugget before it is used up.)

Density independent factors:

- *Death:* Carrots die if older than age 20 or have negative energy. They also die with some probability according to soil water level and temperature. To implement this probability, each carrot is assigned a “survival” value between 0 and 1 for each parameter. At each parameter value, threshold values are calculated according a function of desired behavior; carrots with survival values above the threshold die.
    - Temperature (slider): Carrots grow best at 60-70 degrees. Thresholds are determined by a Gaussian distribution centered around 65 degrees, so all carrots survive at 65 degrees, and survival rates drop off as the temperature gets hotter or colder. The actual equation used: [threshold = exp(-(temp-65)^2/1000)].



    - Soil water levels (slider): Though soil water is necessary for growth, too much water in the soil deprives carrots of oxygen, effectively drowning them. Thresholds are determined by a distribution that drops off at high soil water levels. The actual equation used: [threshold = 1-exp((water-102)/10)].



GRAPH:

The graph shows the current carrot population number (line 1), and the slider values: sunlight (line 2), temperature (line 3), soil water level (line 4), and soil nutrient level (line 5). Though the sliders values also appear on the sliders, having them on the graph makes changes and their correlations to population number easier to track.

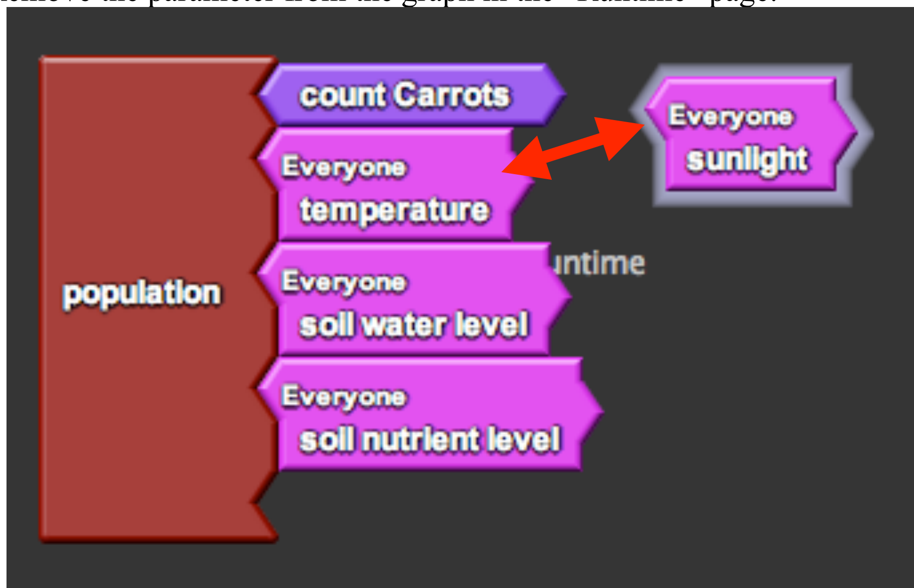
Possible Modifications:
The intended purpose of the many parameters in this model is to allow students to have many possible hypotheses that they could choose from to test, giving them both a sense of individuality as well as an understanding of the scientific process. However, should you choose to use this model instead with the focus of teaching abiotic factors and population growth, it is possible to modify the project to focus on any particular parameter.

To remove a parameter:

1) Disconnect the slider in the "Everyone" page:

2) Remove the setter block from "reset default values" in the "Setup" page and place it under "setup" instead:

3) Remove the parameter from the graph in the “Runtime” page:



Suggested Lesson Guide:

Part 1: Discussion

1 What makes up an ecosystem?
   a) Brainstorm a list on the board.
      - This list can help tie together concepts when we return to it in future lessons.
   b) Highlight ***abiotic elements*** in the list.

2 Introduce the day's activity.
   a) Starting from basics (abiotic factors and producers), build up as we progress.

3 What kinds of abiotic factors affect carrot growth?
   a) Brainstorm a second list on the board.
   b) Introduce the idea of ***density dependence*** versus ***density independence***. Which abiotic factors are density dependent? Which factors are density independent?

4 Explain the idea of using a ***model*** of carrot growth in a ***simulation*** to investigate the effects of these factors.
   a) Why are modeling and simulation useful tools for scientific inquiry?
      - Use this to gauge initial student understanding and get students thinking about why they are doing these activities.
   b) Note that there are six factors in the model that we can investigate: physical crowding, sunlight, temperature, soil water level, soil nutrient level, soil nutrient distribution.

Part 2: Activity

1 Introduce driver/navigator roles (should you choose to use them).
   a) Assign pairs.
   b) Get the students on the computers and open the file.

2 Demo the model first with the default conditions and observe the graph.
   a) Discuss ***exponential*** versus ***logistic growth***.
   b) Note that under default condition the model runs with only physical crowding in play. Discuss the idea of ***limiting factors*** and ***carrying capacity***.

3 Give time for students to play with the model, explore the user interface, change sliders, etc. Explain that they will be performing an experiment on the model and see if they can develop any hypotheses based on preliminary observations.
   a) Hopefully students will get a sense that there are too many variables if they vary all of the sliders at once and understand that they should hone their investigation to just one or two variables.
   b) If the students do not narrow their search to one or two variables, do not force them to do so; instead, keep closer track of them and challenge them to defend their observations.

4 Demo one of the factors (sunlight is the simplest).
   a) Vary the sliders by typing in the values (this trick makes it easier to get accurate incremental values).
   b) Observe the changes in the graph and draw conclusions.
   c) Point out that you can save the graphs in the program.

5 Begin worksheet activity.
   a) If students finish one factor easily, ask them to either investigate another or two in combination. Though the factors themselves are independent in the model, they may have different effects on carrot growth when combined (ex. There will only be one limiting factor, depending on which is the most limited.) They may also have additive or opposing effects.

Part 3: Debrief

1 Ask students to present which factor(s) they investigated, whether they thought it was density-dependent or density-independent, and if they were right or not.
   a) Ask them to support their conclusions based on the data that they collected.
   b) For factors like temperature and soil water level, which have interesting behaviors, see if students could predict what the actual model behavior was.

2 Discuss the model.
   a) How can we tell if a model is accurate or inaccurate?
   b) In what ways was it accurate to real life?
   c) In what ways was it inaccurate? One inaccuracy is that the factors are all interrelated (sunlight affects temperature, etc.)
   d) Is the model still valuable even with inaccuracies?

Student Worksheet:

LESSON 1: ECOSYSTEM MODELING

Create a hypothesis about one or more abiotic factors. Does the factor act in a density-dependent or density-independent (or both) fashion? Predict the carrot's growth behavior as you change the value of the factor. If the factor is density-dependent, at what density does it become a limiting factor? If the factor is density-independent, how well do carrots survive under different values?

Hypothesis:

Collect data from the model. You may save the graphs on the computer or create different ones on graph paper provided.

Observations:

Confirm or contradict your hypothesis. Use data from your observations to support your conclusion. Do you think that this conclusion is valid in real life? Why or why not?

Conclusion:

LESSON 2: VANTS (or termites)

Goals:
The goal of introducing VANTS in the second lesson is to allow students to become familiar with using StarLogo blocks to program. In VANTS, students basically build from scratch, which allows them to build confidence in their own programming abilities (no abstractions). Learning the basic commands and setup in StarlogoTNG during VANTS will also allow students to gain a basic understanding that will make the next activity, programming a rabbit, easier to implement. Students can see large changes arise from simple changes in the program. Additionally, VANTS allows students to be creative, explore, and produce their own individual products. If you choose, you may wish to replace VANTS with the termites activity to focus more on the idea of decentralized systems, or do both activities.

Biology Concepts:
- Agent-based modeling
- Decentralized systems and emergent behaviors
- Systematic changes

StarLogoTNG Programming Concepts:
- How to communicate with the computer via commands
- Block canvas, mini map, block drawer organization and exploration
- Breed editing
- Setup/ Forever blocks
- Clear all/ Clear everyone
- Create agents
- Procedures
- Forward/ Back/ Left/ Right
- Stamp/ Color
- If/ Ifelse/ Boolean Logic
- Blocks exploration (yank, stomp, build, dig, etc.)

Activity:
Refer to the VANTS (or termites) activities for details.

LESSON 3: PROGRAM A BUNNY (2 sessions)

Goals:
The goal of this programming project is for students to gain an understanding for how to build their own models. The program is relatively simple, and the activity should reinforce understanding of block commands that they already know and introduce them to new ones that they don't. Additionally, the procedures that the rabbits are used in later models by various species, so learning them gives the students a better general understanding of how those models work, even though the later activities begin with the species preprogrammed. This activity is long and will likely take 2 sessions to complete.

Biology Concepts:
- Producers and primary consumers
- Predator-prey oscillations
- Food chain vs. food web
- Energy flow and loss/ conversion efficiency
- Procedures of the rabbit model (movement, growth, consumption, sense, reproduction, death)

StarLogoTNG Programming Concepts:
- Scatter agents
- Random
- Collisions
- Variables
- Die
- Hatch
- Smell
- Creating graphs

Materials:
- Starter code: *3-rabbits.sltng*
- Student worksheet
- Projector/ computer for demo
- Blackboard/whiteboard/large paper for brainstorming

About the Model:
The starter code contains a simplified version of the carrot ecosystem where the only limiting factor is space and carrots grow up to carrying capacity. Once the rabbits are programmed, the rabbits and carrots exhibit classic oscillations in population size typical to predator-prey interactions. The following explanations describe the completed solution project.

SETUP:
The model begins with 100 "carrots" and 40 "rabbits" scattered around Spaceland. Carrots begin with a random age (1-10) and random energy (1-20), while rabbits begin with a random age (1-10) and random energy (1-50).

FOREVER:
Carrot procedures:

- Increase age +1
- Update visual height
- Produce energy +2 (if fewer than 10 carrots in radius 10)
- Reproduce (if energy > 20)
- Die (if age > 50)

Rabbit procedures:

- Turn toward any nearby carrots if possible and move forward 1 step.
- Increase age +1
- Decrease energy -0.5
- Reproduce (if age > 15 and energy > 15)
- Die (if age > 40 or no energy)
- Eat carrots during collisions (gain 50% of the carrot's energy)

GRAPH:
The graph shows the current carrot population divided in half (line 1) and the current rabbit population (line 2). The reason the carrot population is scaled is for an easier side-by-side comparison with the rabbit population.

Possible Modifications:
This lesson is structured such that each step teaches a new element to programming in StarLogoTNG. It is geared toward helping students understand the process of model building and giving confidence in their abilities to build such models.

However, if time is limited, you may choose to pre-program certain elements and focus on others. Procedures that you pre-program may be saved on the "Everyone" page. If time is *really* limited, while not recommended, you may still choose to forego the programming altogether and just use the model in the solutions project to observe predator-prey interactions.

Suggested Lesson Guide:

Part 1: Discussion

1 Review concepts from Lesson 1.
   a) Assign driver/navigator pairs, get the students on the computers, and open the file.
   b) Run the model with only the carrots. Is this behavior the same as last time? We have taken out all the factors except for crowding. Is this a reasonable simplification?
   c) Return to the list from Lesson 1 on what makes up an ecosystem. Note that we are now going to address primary consumers.
   d) Ask students to hypothesize how the growths of both populations will change over time when rabbits are added to the system.

2 How do we model a rabbit? What actions do rabbits perform on a regular basis?
   a) Brainstorm a list on the board.
   b) Note the ones that we will include in our model: aging, losing energy (metabolizing), eating carrots, moving, reproducing, dying.

Part 2: Activity

1 Begin guided programming worksheet activity.
   a) Introduce each section of the worksheet with a discussion of how to the students would program this feature. Though the students may come up with suggestions that are either incompatible or too difficult to implement in the program, it is still good to get them into the mindset of being a programmer.
   b) Perform each of the steps on the projector and discuss the changes as a class before asking students to implement individually. This keeps the class at a somewhat even pace and covers the nuances that students may miss.
   c) If you feel students have a firm grasp on how to proceed, you may wish to let them continue with the worksheet on their own. However, it is recommended that this not be done until after the lesson on variables, which are historically difficult for students to understand. The explanation in the worksheet for variables makes an attempt to deliver a more intuitive understanding, but other analogies may work even better.
   d) Constant feedback from the program after adding blocks as a group will provide ample opportunity to discuss programming concepts and debugging solutions. As an example, click “setup” immediately after creating a new breed and ask students where the new characters are. Some may notice that they don’t exist because they haven’t created them yet! This type of critical thinking and debugging logic will help students understand their own code as they progress.

2 Once the rabbits are complete, ask the students to observe the graph.
   a) Was this the behavior between carrots and rabbit that was predicted?
   b) Are the population cycles in sync? Why not?

3 If some students finish before others, challenge them to program something new into their model. This does not have to be anything too realistic, such as making their rabbits speak, but can help foster a sense of ownership and a higher level of comfort with the software.

Part 3: Debrief

1 Ask students what they learned from the activity (answers can be about the process of modeling, programming, StarLogo, ecosystems, etc.).

2 Discuss the model.
   a) Which other rabbit actions that we brainstormed could be easily added to this model? How would you implement them in code?
   b) What would happen if we added other elements into the model?

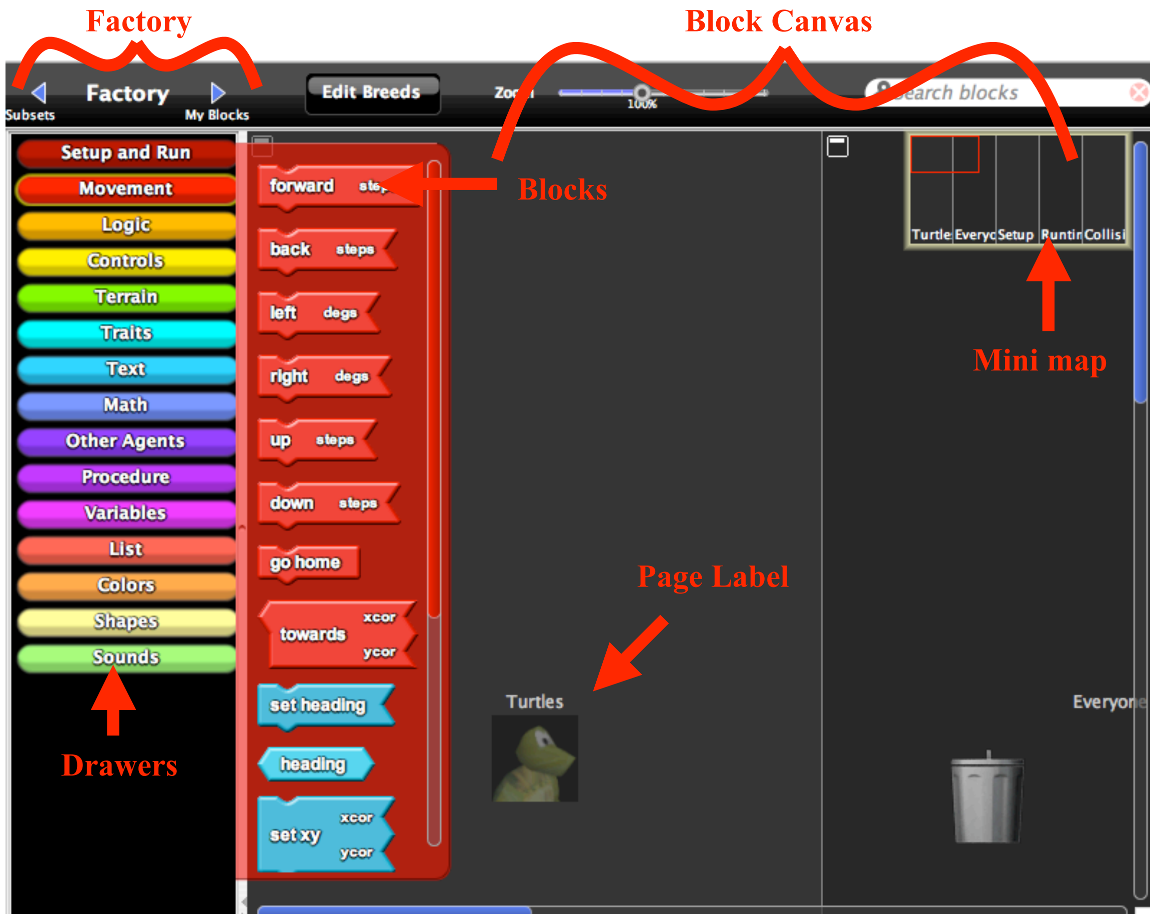
Student Worksheet:

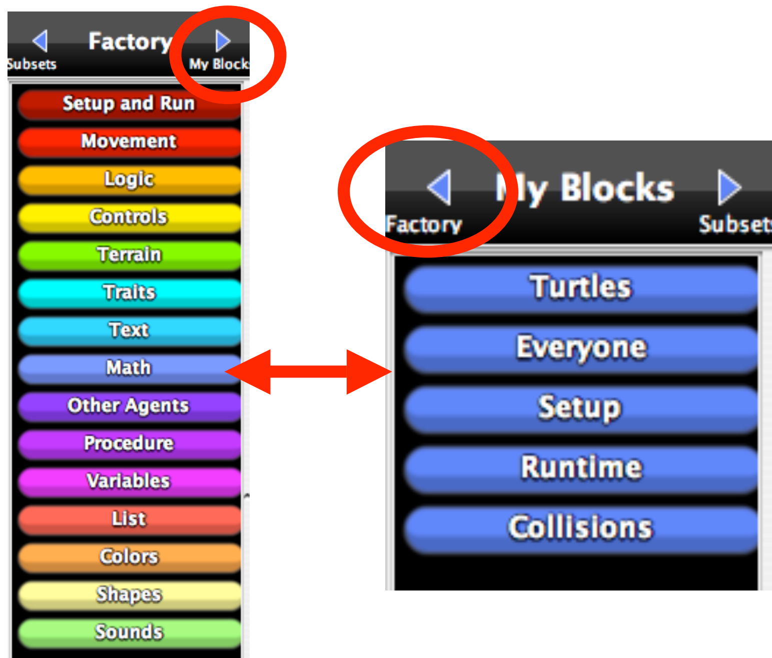
LESSON 3: PROGRAM A BUNNY

Part 0: StarLogo vocabulary and navigation.

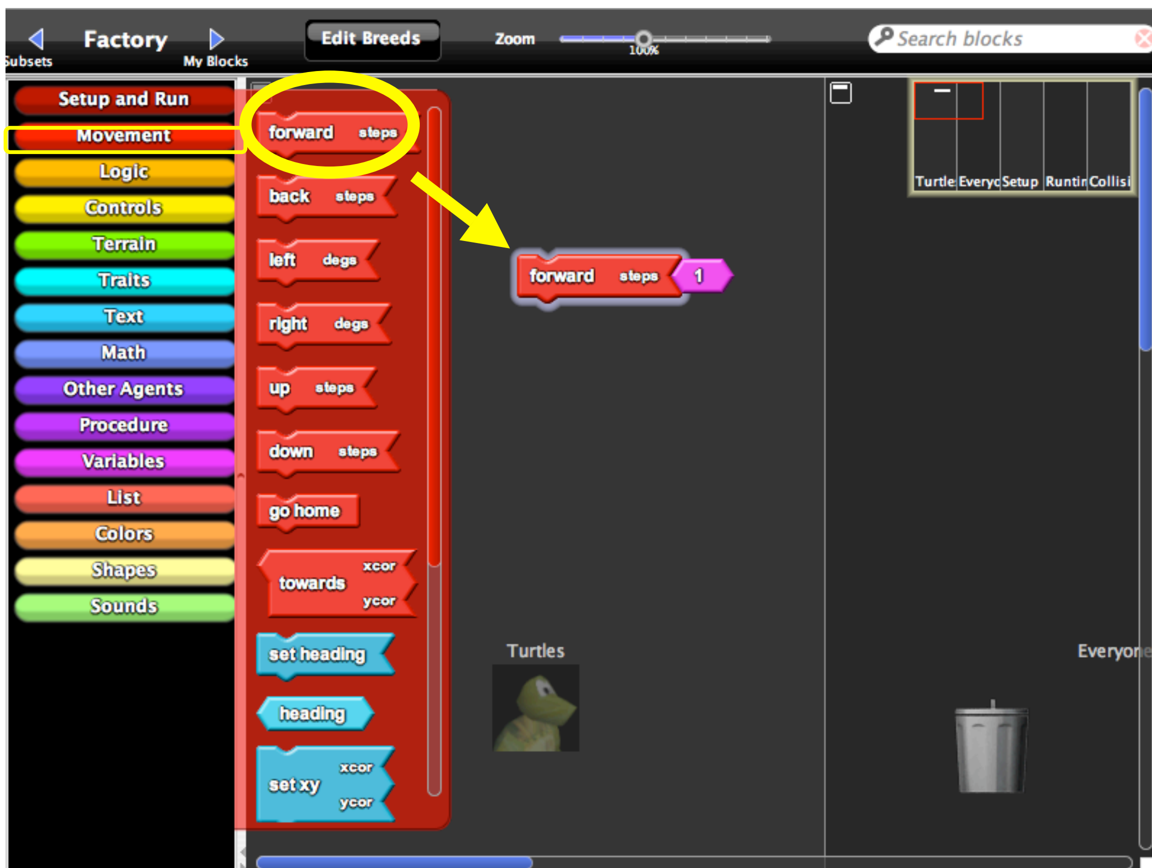
This is the vocabulary that will be used in this activity. You may reference this diagram to help you find blocks while you program.

- In StarLogoTNG, all agents are assigned a certain ***breed***.
- Commands to control the breeds are called ***blocks***.
- The programming area of StarLogoTNG is called the ***block canvas***.
- On the canvas there are separate sections called ***pages*** to help organize your code.
- You can navigate these pages through the ***mini-map*** in the upper right hand corner.
- The colored tabs on the left hand side are called ***drawers***. You can click on them to open and close them. Here is where you find blocks that you want to use, which you drag from the drawer onto the canvas.
- Drawers are organized under either the ***Factory*** or ***My Blocks***. You can click on the right arrow next to Factory that says "My Blocks" to get to My Blocks, and you can click on the left arrow next to My Blocks that says "Factory" to return to the Factory.





Here is an example of directions you will see:
"Take the **"forward"** ***block*** from the **"Movement"** ***drawer*** under the ***Factory***. Drag it onto the **"Turtles"** ***page*** on the ***block canvas***." This translates to the following action:

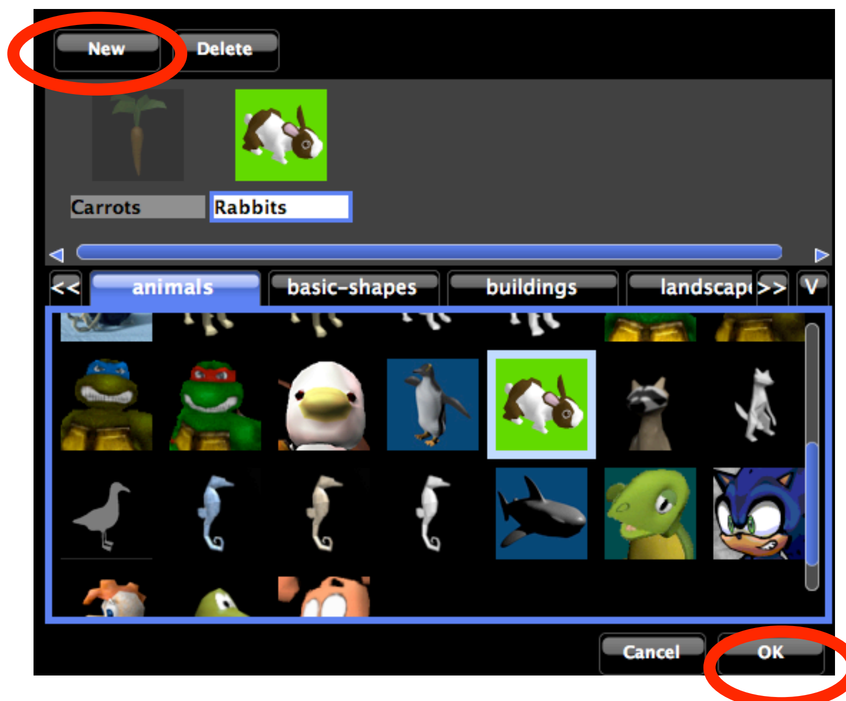


Part 1: Creating a rabbit breed.

1) Adding a breed.

   a) Click on the button next to the Factory that says "**Edit Breeds**." This should pop open a window called the Breed Editor.

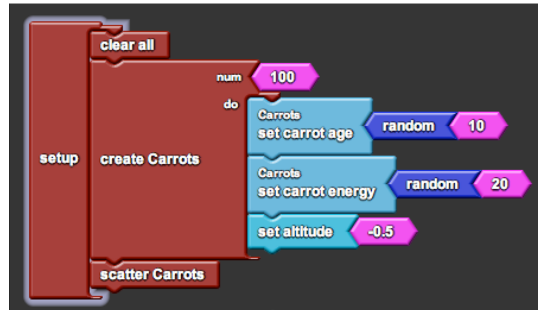
   b) Click on "New" to create a new breed and give it a name (i.e. "Rabbits"). Choose a shape for the breed, then click "OK." Do you notice anything different?



A new page has been automatically created for the new breed, along with a drawer in **My Blocks** and various blocks within that drawer.

2) Setting up the breed.

a) Locate the **"Setup"** page on the block canvas and find the **"setup"** stack. This stack contains all the commands that will be executed when you press the **"setup"** button in the Runtime window.



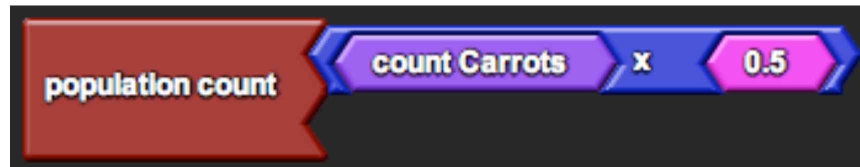
b) Find the **"create Rabbits [num, do]"** block from the **"Rabbits"** drawer under **My Blocks**. This block creates [num] number of rabbits and asks them to do the commands listed under [do].

c) Drag it onto the **"Setup"** page on the block canvas and hook it under **"scatter Carrots"** in the **"setup"** stack. Change the number **"10"** to read **"40."**

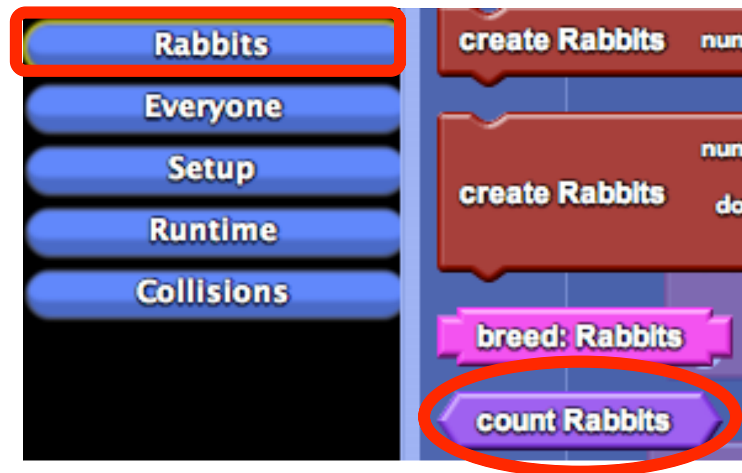
d) Repeat the process for the **"scatter Rabbits"** block. This block scatters the rabbits to random locations around Spaceland. Now the program will create 40 rabbits and scatter them when you press **"setup."**

3) Graphing

a) Locate the **"Runtime"** page on the block canvas and find the **"population count"** block. This block corresponds to the graph in the Runtime window. It currently graphs only the number of carrots, scaled by a half (this is so that it will be easier to compare carrots and rabbits later on, since there are many more carrots than rabbits in the model).



b) Find the **"count Rabbits"** block from the **"Rabbits"** drawer under **My Blocks**. This block returns the total number of rabbits that are alive in Spaceland.



c) Drag it onto the "**Runtime"** page on the block canvas and stick it in the socket of the **"population count"** block. Now the graph in the Runtime window should show two lines - one for carrots and one for rabbits.

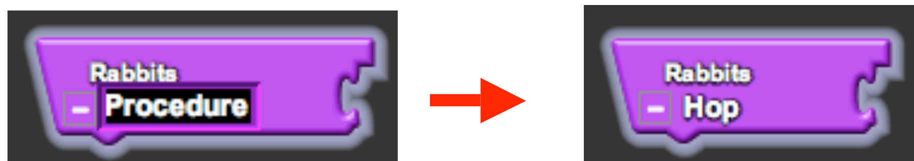
After you make each change to the model, try running it. Observe the graph to see how each new rabbit behavior affects the population dynamics.

Part 2: Simple rabbit movement (procedures).

1) Creating a procedure declaration.

   a) Find the **"Procedure"** block from the **"Procedure"** drawer under **Factory**.

   b) Drag it onto the **"Rabbits"** page and rename it **"Hop**."

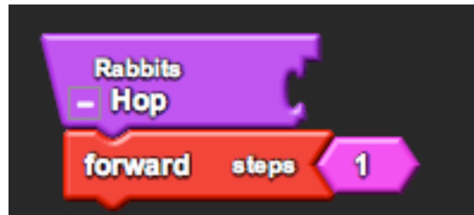


   This is a ***procedure declaration block***. Underneath the block, we can stack commands that define the procedure. Now whenever the procedure is called, the rabbit will execute the entire list of commands in order.

   For instance, say our rabbit "hops" by moving forward, wiggling its nose, looking around, sitting down, and munching on grass. Think of the procedure as a shortcut command: Let's say our rabbit hops *a lot*. Rather than typing the entire sequence of commands each time, we only have to type "hop" and it will know to do the entire list of actions. Much shorter, right?

   c) For now our rabbits are lazy and only move forward. Find the **"forward"** block from the **"Movement"** drawer under **Factory**. This block makes the rabbit move forward [steps] number of steps.

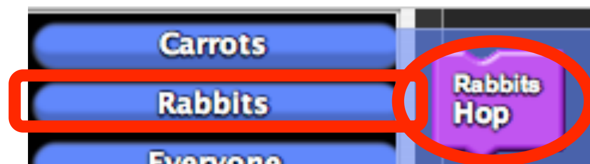
d) Drag it onto the **"Rabbits"** page under **"Hop."** Now your "hop" procedure is defined as a simple forward movement of step size 1.



2) Calling the procedure.

a) Locate the **"Runtime"** page on the block canvas and find the **"forever"** block. This block contains all of the commands that will be run over and over again when you press the **"forever"** button in the Runtime window.

b) Find the **"Hop"** block from the **"Rabbits"** drawer under **My Blocks**.



This is a ***procedure call block***. It is generated automatically when you create a procedure using a procedure declaration block like you did in the previous step. This declaration block was merely a definition, whereas this call block is the actual command. Use this block when actually asking the rabbit to "hop."

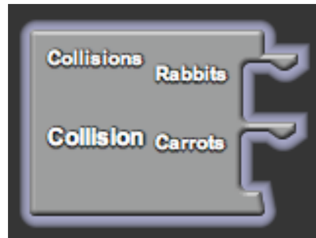
c) Drag it onto the **"Runtime"** page and hook it under the **"Rabbits"** hook in the **"forever"** block. Now the rabbits will hop when you press **"forever."**

Part 3: Rabbits eating carrots (collisions).

1) Defining a collision.

   a) Find the **“Collision”** block between ***Rabbits*** and ***Carrots*** from the **“Rabbits”** drawer under **My Blocks**.

   b) Drag it onto the **“Collisions”** page.



   This is a ***collision block***. The program automatically checks for collisions between rabbits and carrots. In the event of a collision, each of the colliding agents performs the command listed under its hook.

   c) When our rabbits collide with carrots, they eat the carrots. Visually, we see the carrots die. Find the **“die”** block from the **“Logic”** drawer in the **Factory**.

   d) Drag it onto the **“Collisions”** page and hook it under the **“Carrots”** hook in the **“Collision”** block. Now any carrot that collides with a rabbit will die.

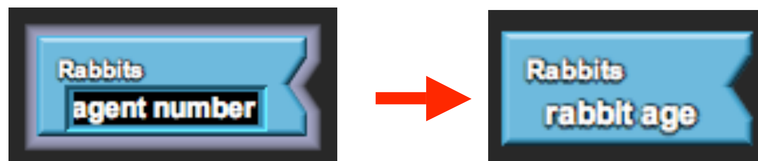
Part 4: Rabbit age and energy (agent variables).

1) Creating a variable declaration.

   What is a variable? A variable is simply some value that the program wants to keep track of. As part of the name "variable," its value may or may not vary. There are two types of variables: an ***agent variable*** belongs to only that particular agent, while a ***shared variable*** is shared and accessed by everyone. In most cases the variable is some trait, such as color, height, size, etc., but in other cases the variable could simply keep tabs on some state, such as if it's raining or not.

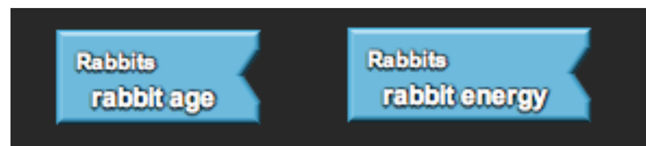
   a) Find the **"agent number"** block from the **"Variables"** drawer in the **Factory**.

   b) Drag it onto the **"Rabbits"** page and rename it **"rabbit age**."



   This is a ***variable declaration block***. You can think of the variable declaration block as clearing a chunk of storage space for the variable. For instance, say we are keeping track of our rabbit's age. Without a declaration, the program does not even know the rabbit has an age and consequently will not try to save space for it. The agent number is a type of agent variable that keeps track of a number value.

   c) Repeat the process for **"rabbit energy**.**"** Now the program knows that rabbits have both an age and an energy.



2) Setting the variable initial value.

   a) Relocate the **"Setup"** page and find the **"setup"** stack from Part 1 (2d).

b) Find the **“set rabbit age”** block from the **“Rabbits”** drawer in **My Blocks**.

This is a ***variable setter block***. It is generated automatically when you create a variable using a variable declaration block like you did in the previous step.

Though we’ve declared that the rabbit has an age, the program still does not have a clue how old the rabbit is. If you ask the program what the rabbit’s age is, it will either go crazy or tell you some nonsensical value. We use a setter block to set the value, effectively telling the computer to save the value in its storage space.

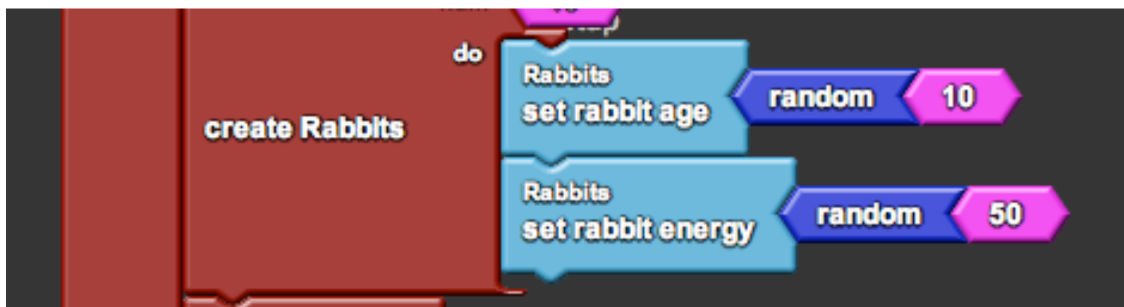
c) Drag it onto the **“Setup”** page and hook it under the **“do”** hook in the **“create Rabbits”** block. Type the number **“10”** and press enter. (Connect the blocks if they don’t automatically connect.)

d) Repeat the process for **“set rabbit energy”** and the value **“50**.**”**

e) While the rabbits could begin with a set age, we want to insert a bit of chance into the model. To do this, find the **“random”** block from the **“Math”** drawer in the **Factory**. This block picks a random number between 1 and the input. For instance, random 10 will return a value between 1 and 10.

f) Drag it onto the **“Setup”** page and stick it between **“set rabbit age”** and **“10**.**”**

g) Repeat the process for **“set rabbit energy”** and the value **“50**.**”** Now each rabbit will begin with some random age between 1 and 10 and some random energy between 1 and 50.



Part 5: Rabbit growth and death (logic).

1) Aging and energy loss.

   With age and energy variables, we can more accurately represent the life of a rabbit, which does not run around forever. As rabbits grow, they get older and use energy.

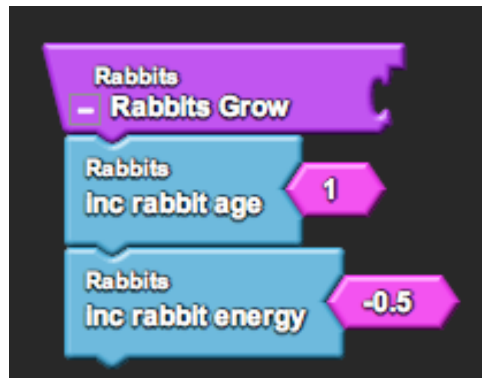
   a) Find the **“Procedure”** block from the **“Procedure”** drawer under **Factory**.

   b) Drag it onto the **“Rabbits”** page and rename it **“Rabbits Grow**.”

   c) Find the **“inc rabbit age”** block from the **“Rabbits”** drawer in **My Blocks**. This block increases the rabbits age by a given amount. Type the number **“1”** and press enter. (Connect the blocks if they don’t automatically connect.)

   d) Drag it onto the **“Rabbits”** page under **“Rabbits Grow**.”

   e) Repeat the process for **“inc rabbit energy”** and the value **“-0.5**.” Now the “rabbits grow” procedure is defined as an increase in rabbit age by 1 and a decrease in rabbit energy by 0.5.



2) Energy gain through eating.

   Now that the rabbit loses energy, it must have some way to gain the energy back. We already programmed the rabbits to “eat” carrots by making the carrots die upon collision. However, when the rabbits eat the carrots, they should also gain energy.

   a) Relocate the **“Collisions”** page and find the **“Collision”** block from Part 3 (2d).

   b) Find the **“inc rabbit energy”** block from the **“Rabbits”** drawer in **My Blocks**.

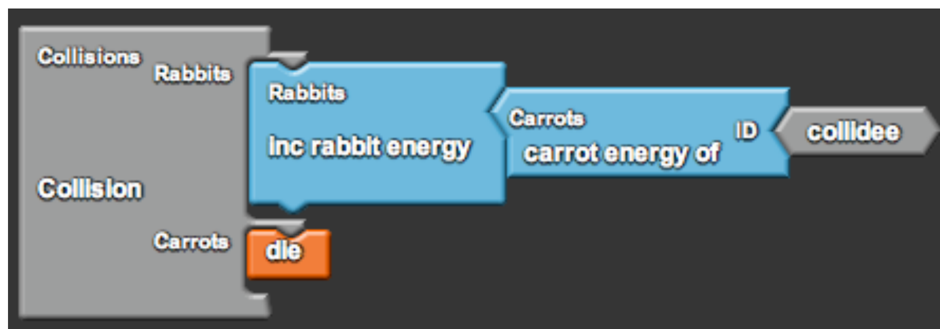
   c) Drag it onto the **“Collisions”** page and hook it under the **“Rabbits”** hook in the **“Collision”** block.

d) Find the **"carrot energy of [ID]"** block from the **"Carrots"** drawer in **My Blocks**. This block allows the rabbit to inquire about the energy that the carrot has. [ID] is the ID number of the carrot that the rabbit is inquiring about.

e) Drag it onto the **"Collisions"** page and stick it in the socket of the **"inc rabbit energy"** block.

f) Find the **"collidee"** block from the **"Other Agents"** drawer in the **Factory**. This block returns the ID number of the carrot that the rabbit collided with.

g) Drag it onto the **"Collisions"** page and stick it in the **"ID"** socket of the **"carrot energy of"** block. Now whenever a rabbit and carrot collide, the rabbit will increase its energy by the amount of energy that that carrot had.

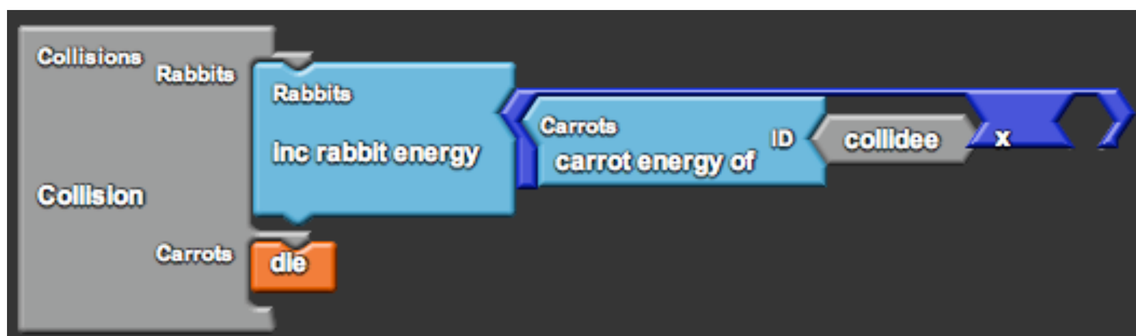


3) Energy efficiency.

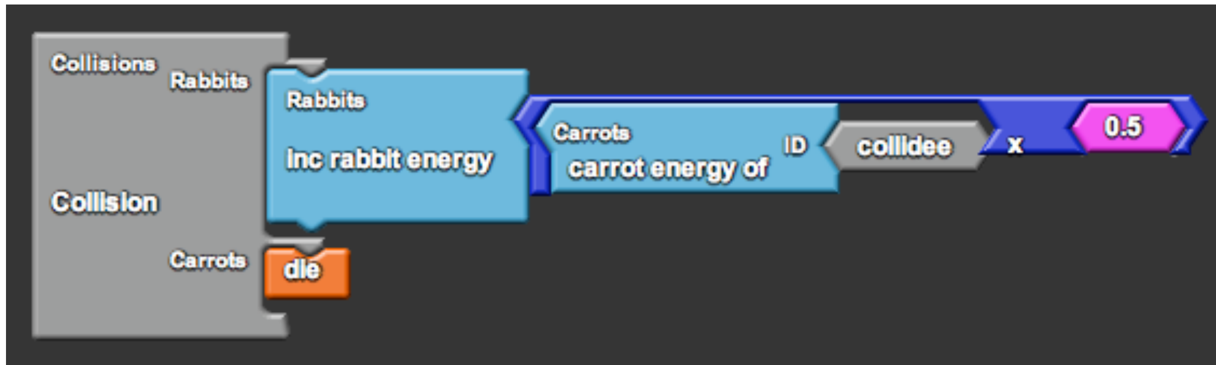
In real life there is some amount of energy lost in the form of heat during consumption at each stage of the food chain. Though this energy is on the order of 90%, we will model it in our system as 50%. Hence, we want to modify the amount of energy that the rabbit gains to be only half of the energy that the carrot had.

a) Find the **"x"** block from the **"Math"** drawer in the **Factory**. This block multiplies two numbers.

b) Drag it onto the **"Collisions"** page and stick it between **"inc rabbit energy"** and **"carrot energy of [ID]."**



c) Type the number **"0.5"** and press enter. Attach the block in the empty socket of the **"x"** block. Now the rabbits only gain half of the carrot energy upon collision.



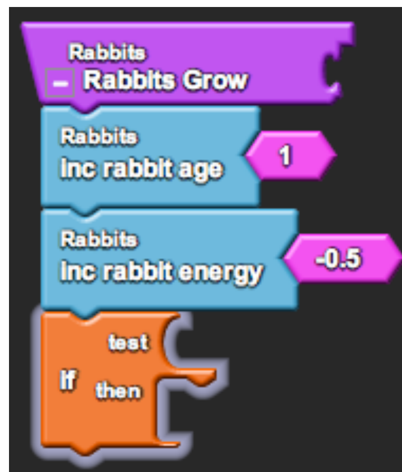
4) Death.

Finally, rabbits die of old age and weakness. We will program death into the system by checking if it is the rabbit's time to die.

a) Relocate the **"Rabbits"** page and find the **"Rabbits Grow"** procedure from Part 5 (1e).

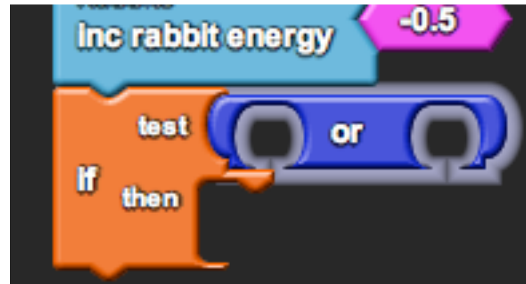
b) Find the **"if [test, then]"** block from the **"Logic"** drawer in the **Factory**. This block checks if a condition is true or false and performs the action specified *only* if the condition is true.

c) Drag it onto the **"Rabbits"** page under the **"inc rabbit energy"** block in **"Rabbits Grow**."



d) When do we want the rabbits to die? In our model we will tell our rabbits to die if they are either too old *or* have no energy left. Find the **"or"** block from the **"Logic"** drawer in the **Factory**. This block will allow us to check two conditions and returns true if *at least one* of them (one *or* the other) is true.

e) Drag it onto the **“Rabbits”** page and stick it in the **“test”** socket of the **“if”** block.



f) Find the **“>”** block from the **“Math”** drawer in the **Factory**.

g) Drag it onto the **“Rabbits”** page and attach it in the first socket of the **“or”** block.

h) Find the **“rabbit age”** block from the **“Rabbits”** drawer in **My Blocks**. This block returns the current age of the rabbit.

i) Drag it onto the **“Rabbits”** page and attach it in the first socket of the **“>”** block. Type the number **“40”** and press enter. Attach the block in the second socket.

j) Repeat the process for **“<”** and the values **“rabbit energy”** and **“0.”** Now the rabbits check if they are either older than 40 (rabbit age > 40) *or* have no energy left (rabbit energy < 0).

k) Though the rabbits check for the death condition, they must also actually die. Find the **“die”** block from the **“Logic”** drawer in the **Factory**.

l) Drag it onto the **“Rabbits”** page and hook it under the **“then”** hook in the **“if”** block. Now the rabbits check if they should die, and if they should, they do.

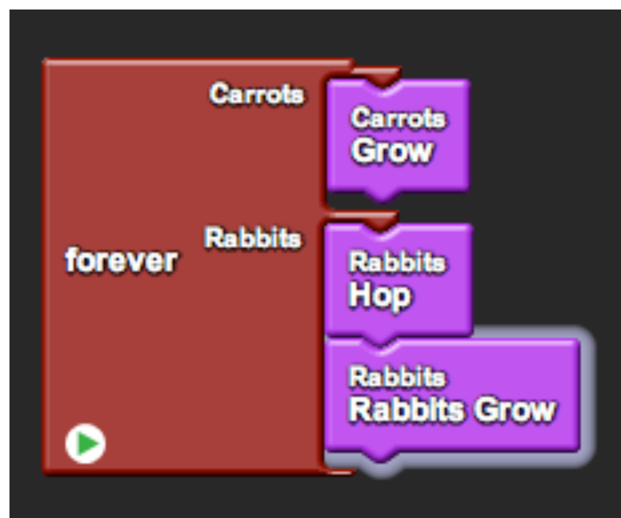
5) Calling the procedure.

You may have noticed that the rabbits still do not grow. Just like with rabbit "hop," we have only defined the procedure so far. We need to actually ask the rabbit to perform the "rabbits grow" procedure before we see any results.

a) Relocate the **"Runtime"** page and find the **"forever"** block from Part 2 (2c).

b) Find the **"Rabbits Grow"** block from the **"Rabbits"** drawer under **My Blocks**.

c) Drag it onto the **"Runtime"** page under the **"Hop"** block in **"forever**.**"** Now the rabbits will grow when you press **"forever**.**"**



Part 6: Rabbit reproduction (hatch).

1) Hatching an offspring.

   If all of the rabbits die over time, how do we sustain the population? The answer is that the rabbits reproduce. For the simplicity of this model, our rabbits reproduce *asexually*. How might you program rabbits that reproduce sexually?

   a) Find the **"Procedure"** block from the **"Procedure"** drawer in the **Factory**.

   b) Drag it onto the **"Rabbits"** page and rename it **"Reproduce**."

   c) Find the **"hatch [do]"** block from the **"Logic"** drawer in the **Factory**. This block hatches a new rabbit that is an exact replica of its parent. Upon hatching it will do the commands listed under [do].

   d) Drag it onto the **"Rabbits"** page under **"Reproduce."** Now the "reproduce" procedure is defined as the rabbit creating an exact replica of itself.

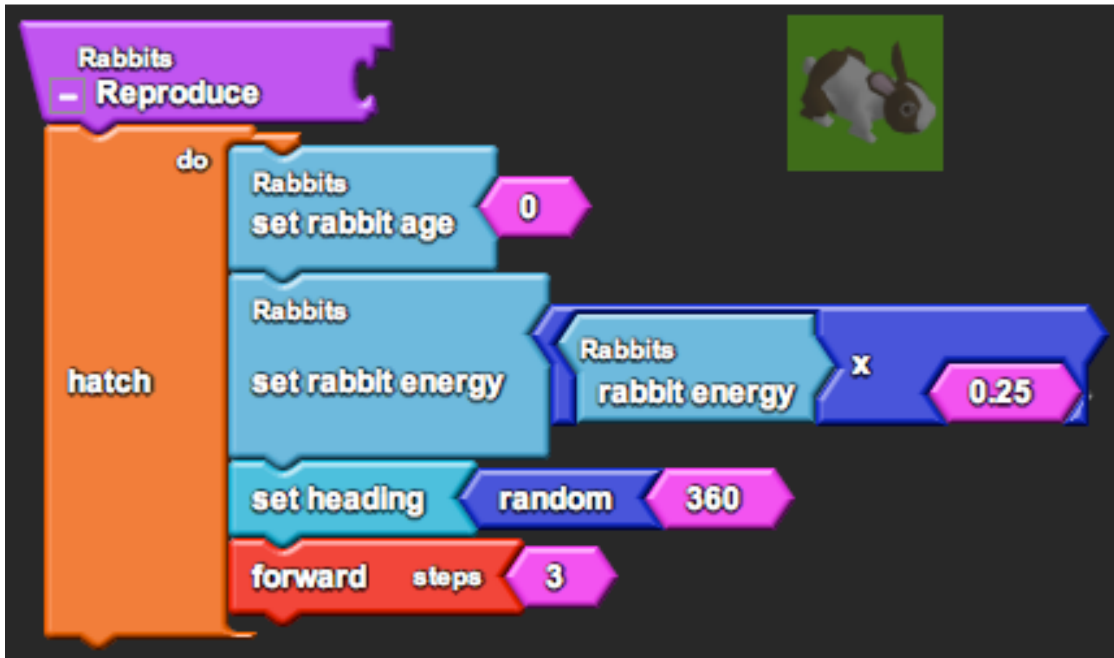
2) Altering the offspring's traits.

   We know that offspring aren't born to be clones of their parents, so we must alter the offspring's traits upon hatching. In particular, we will as the offspring to be newborn (age = 0), with less energy (energy = 25% parent energy), and a slight distance away from the parent.

   a) Find the **"set rabbit age"** and **"set rabbit energy"** blocks from the **"Rabbits"** drawer in **My Blocks**. Find the **"set heading"** and **"forward"** blocks from the **"Movement"** drawer in the **Factory**.

   b) Drag them onto the **"Rabbits"** page and hook them sequentially under the **"do"** hook in the **"hatch"** block.

   c) Attach the following values for each (you know how to do this!):
      i) **"0"** in the socket of **"set rabbit age**."
      ii) **"rabbit energy x 0.25"** in the socket of **"set rabbit energy**."
      iii) **"random 360"** in the socket of **"set heading**."
      iv) **"1"** in the socket of **"forward**."



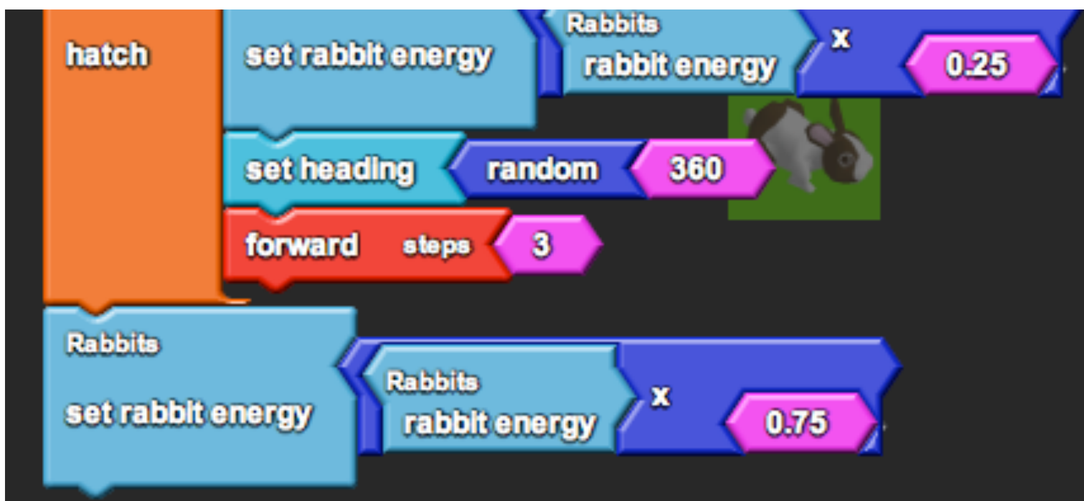
3) Losing energy.

   Since the parent rabbit shared 25% of its energy with its offspring, it decreases its own energy to 75%.

   a) Find the **"set rabbit energy"** block from the **"Rabbit"** drawer in **My Blocks**.

   b) Drag it onto the **"Rabbits"** page under the **"hatch"** block in **"Reproduce**." Note that the block must be *outside* the **"hatch"** block because it is the parent rabbit that is setting its energy and not the offspring.

   c) Attach **"rabbit energy x 0.75"** in the socket of the **"set rabbit energy"** block. Now the parent rabbit loses 25% of its energy when it reproduces.



4) Calling the procedure.

As before, we need to actually ask the rabbit to perform the "reproduce" procedure before we see any results. However, we don't want the rabbits to reproduce constantly, rather only when they are sexually mature and have enough energy.

a) Since reproduction is part of the rabbit growth cycle, we will put the procedure under the **"Rabbits Grow"** procedure. Relocate the **"Rabbits"** page and find the **"Rabbits Grow"** procedure from Part 5 (5c).

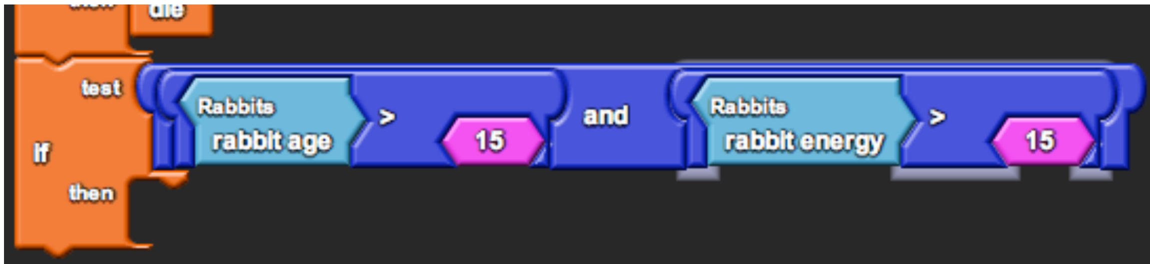
b) Find the **"if [test, then]"** block from the **"Logic"** drawer in the **Factory**.

c) Drag it onto the **"Rabbits"** page under the **"if [test, then]"** block in **"Rabbits Grow."**

d) When do we want the rabbits to reproduce? In our model we will tell our rabbits to reproduce if they are both of age *and* have enough energy. Find the **"and"** block from the **"Logic"** drawer in the **Factory**. This block will allow us to check two conditions and returns true if *both* of them (one *and* the other) is true.

e) Drag it onto the **"Rabbits"** page and stick it in the **"test"** socket of the **"if"** block.

f) Attach **"rabbit age > 15"** and **"rabbit energy > 15"** in the two sockets of the **"and"** block. Now rabbits check if they are both older than 15 (rabbit age > 15) *and* have enough energy (rabbit energy > 15).



g) Find the **"Reproduce"** block from the **"Rabbits"** drawer in **My Blocks**.

h) Drag it onto the **"Rabbits"** page and hook it under the **"then"** hook in the **"if"** block. Now the rabbits check if they should reproduce, and if they should, they do.

Part 7: Rabbit senses and smart movement (smell).

1) Rabbits smell.

   Now that we have a fully functioning rabbit population, we can make the rabbits smarter. Rabbits can track down the carrots by sensing their proximity.

   a) Relocate the **“Rabbits”** page and find the **“Hop”** procedure from Part 2 (1d).

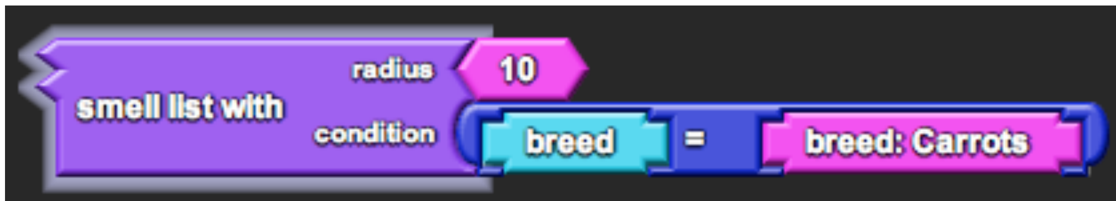
   b) Find the **“smell list with [radius, condition]”** block from the **“Other Agents”** drawer under the **Factory**. This block returns a list of ID numbers of agents that are in the radius [radius] of the rabbit that satisfy the condition [condition].

   c) Drag it onto the **“Rabbits”** page next to the **“Hop”** procedure. We’re not going to use it yet, but let’s investigate how to set it up.

   d) Type the number **“10”** and press enter. (Connect the block to [radius] if they don’t automatically connect.)

   e) Find the **“=”** block from the **“Logic”** drawer under the **Factory**. Find the **“breed”** block from the **“Traits”** drawer under the **Factory**. Find the **“breed: Carrots”** block from the **“Carrots”** drawer under **My Blocks**.

   f) Drag them onto the **“Rabbits”** page. Attach **“breed = breed: Carrots”** in the [condition] socket of the **“smell list with”** block. This will return a list of only carrots within a radius of 10 of that particular rabbit.



2) Using the rabbit sense.

   Equipped with the list of all the carrots in the vicinity, the rabbits can track one down rather than move forward blindly, hoping to encounter it. To do this, the rabbit should first check that there is in fact a carrot to chase. If so, then the rabbit should turn to face the carrot and *then* move forward. Otherwise, the rabbit should just continue forward, since there is no reason to deviate from the current path.

   a) Find the **“if [test, then]”** block from the **“Logic”** drawer in the **Factory**.

   b) Drag it onto the **“Rabbits”** page above the **“forward”** block in **“Hop.”**

c) Now let’s use this list we made to make the rabbits turn. Find the **“any? [agent list]”** block from the **“Everyone”** drawer under **My Blocks**. This procedure takes the list and returns true if it contains one or more ID numbers, indicating that there are carrots within the radius 10.

d) Drag it onto the **“Rabbits”** page and stick it in the **“test”** socket of the **“if”** block.

e) Attach the **“smell list with”** blocks that you set up in the **“agent list”** socket of the **“any?”** block.

f) Find the **“turn towards [agent ID]”** block from the **“Everyone”** drawer under **My Blocks**. This procedure takes the ID number of some agent and makes the rabbit turn to face that agent.

g) Drag it onto the “**Rabbits”** page and stick it in the **“then”** socket of the **“if”** block.

h) Create another set of **“smell list with”** blocks.

i) Find the **“first [list]”** block from the **“List”** drawer under the **Factory**. This procedure takes the first value from the list, which here means the ID number of the first carrot that the rabbit sensed.

j) Drag it onto the **“Rabbits”** page and stick the **“smell list with”** blocks in the **“list”** socket of the **“first”** block.

k) Attach **“first”** in the **“then”** socket of the **“if”** block. Now the rabbits should move in such a way that they can chase after carrots.

Part 8: Analysis.

Now your rabbit model is complete. Your rabbits do all of the following:

- Move and chase after carrots.
- Grow older.
- Lose energy to metabolism.
- Gain energy from eating carrots.
- Die if too old or not enough energy.
- Reproduce if old enough and enough energy.

1) Observe the graph of the carrot and rabbit population sizes. Is this the behavior you predicted? How did it change as you added new elements to the rabbit's behavior?

2) How does the carrot population size affect the rabbit population size? Vice versa? Support this with data from the graphs.

3) Why do you think that the populations behave in such a way?

LESSON 4: HUMAN INTERACTIONS

Goals:
The goal of this project is to focus on the question of where humans fit in the ecosystem. Ethical and philosophical questions about what is natural for an ecosystem may become a source of debate, and the discussion generated will be valuable for students to form their own opinions about a topic that has no right or wrong answer. Additionally, the gaming element of the activity should be fun while introducing the concept of how to program keyboard controls and score (should the students want to use the program to make games in the future).

Biology Concepts:
- Secondary consumers
- Small populations
- Overpopulation
- Role of humans

StarLogoTNG Programming Concepts:
- First person gaming
- Keyboard controls
- Score

Materials:
- Starter code: *4-hunter.sltng*
- Student worksheet
- Projector/ computer for demo
- Blackboard/whiteboard/large paper for brainstorming

About the Model:
The starter code contains all the code necessary to run and observe a model of carrots, rabbits, and wolves. The model is written exactly the same as the carrot and rabbit model from Lesson 3 with certain parameters changed. The wolves follow the same code as the rabbits (except that they chase rabbits and not carrots), and the rabbits follow the same code as before (except that they run away from the wolves as well). The code also contains the bullet breed, along with the code associated with bullet movement, which will be used later in programming the hunter. Once the hunter is programmed, it shoots bullets, which update the score if they hit any animals. The following explanations describe the completed solution project.

SETUP:
The model begins with 100 “carrots,” 40 “rabbits,” and 10 “wolves” scattered around Spaceland. Carrots begin with a random age (1-10) and random energy (1-20), rabbits begin with a random age (1-10) and random energy (1-50), and wolves begin with a random age (1-50) and random energy (1-50). There is 1 “hunter,” who begins in the middle of Spaceland, and the camera is set to be the viewpoint of the hunter. The score and the initial number of wolves and rabbits killed are set to 0.

RUN (120 sec):
Carrot procedures:
- Increase age +1
- Update visual height
- Produce energy +2 (if fewer than 10 carrots in radius 10)
- Reproduce (if energy > 20)
- Die (if age > 50)

Rabbit procedures:
- If there are any wolves nearby, turn away and move forward 2 steps (run away).
- Otherwise, face any nearby carrots if possible and move forward 1 step (chase).
- Increase age +1
- Decrease energy -0.5
- Reproduce (if age > 15 and energy > 15)
- Die (if age > 40 or no energy)
- Eat carrots during collisions (gain 50% of the carrot's energy)

Wolf procedures:
- If there are any rabbits nearby, face one and move forward 3.5 steps (chase).
- Otherwise, turn randomly and move forward 1 step (prowl).
- Increase age +1
- Decrease energy -0.25
- Reproduce (if age > 15 and energy > 30)
- Die (if age > 50 or no energy)
- Eat rabbits during collisions (gain 90% of the rabbit's energy)

Hunter procedures:
- Move forward, back, right, left according to keyboard controls.
- Shoot a bullet according to keyboard spacebar control. (Note that "shoot" only hatches a bullet, while the bullet itself travels forward and hits animals.)
- Update score to be [10*wolves killed – 2*rabbits killed].

Bullet procedures:
- Travel forward ±2 degrees.
- If close (within 0.5 radius) to a wolf, mark the wolf as hit by turning it red and wounding it to 0.5 energy. Else, repeat for rabbits. After a hit, the bullet "dies."
- If it is really close (within 2) from the wall, the bullet "dies."
- The bullet repeats this procedure 15 times, giving it a speed of 15. (Note that we didn't use collisions here because collision code is only run after agents complete their code. We want the bullet to check for a hit continuously as it travels.)

GRAPH AND MONITORS:
The graph shows the current carrot population divided in half (line 1), the current rabbit population (line 2), and the current wolf population (line 3). The "rabbits alive," "rabbits killed," "wolves alive," and "wolves killed" monitors are a simple way to keep tabs on the current population dynamics while playing without opening and analyzing the graph.

Possible Modifications:
The game is an open-ended fun activity. Add whatever creative modifications you wish.

Suggested Lesson Guide:

Part 1: Discussion

1 Review concepts from Lesson 3.
   a) Return to the list from Lesson 1 on what makes up an ecosystem. Note that we are now going to address secondary consumers.
   b) Assign driver/navigator pairs, get the students on the computers, and open the file.
   c) Ask each pair to run the model until completion (120 seconds) and note the graph.
   d) Have the class reconvene to discuss what happened. Most of the students will likely observe the population stabilize, but some may notice their ecosystems crash, with both rabbits and wolves (and maybe even carrots) dying out.
   e) Why does this happen? Discuss the idea of small populations. In particular, note that small populations can be volatile, which is a problem to keep in mind when we model. Highlight how chance events play the major role in which populations succeed in stabilizing.
   f) Recall the predator-prey cycles from Lesson 3. How do the wolf, rabbit, and carrot populations interact?

2 Change the balance of the ecosystem.
   a) Ask the students to locate the section on the **"Wolves"** page for **"Wolves Grow"**:

b) Demo changing the wolf energy necessary for reproduction to 15. Then ask the students to do the same in their programs.

c) Run the simulation again and note what happens (wolves overpopulate).

3 What other factors may cause over population of a species?
   a) Brainstorm a list on the board.
      - Ex. Lower age of reproduction, lower energy loss, longer lifespan, etc.
   b) Try to make a distinction between factors that can be changed in the model and factors that may change in real life.

4 What can be done to maintain the population sizes?
   a) Brainstorm a second list on the board.
   b) Note that we will focus on hunting in this activity. Students may or may not feel that hunting is ethical, but you may want to put off this discussion until after the activity. Alternatively, discuss the ethics first and return to it after the activity to see if any new perspectives came up.

Part 2: Activity

1 Begin guided programming worksheet activity.
   a) Note to students the goal of this activity. They will program a first-person hunter that is controlled by keyboard keys, and they are trying to get the maximum score by killing the most wolves and least rabbits. Again, if students object to the ethics, you may want to put off the discussion until after the activity.
   b) You may demo this activity first or just let the students program with the worksheet, depending on how comfortable you feel they are to do so.

2 Once the hunter is complete, let the students play the game a few times (keeping track of their scores) before reeling in for discussion.

Part 3: Debrief

1 Tabulate the high scores.
   a) Ask students to self report their highest scores.

b) Ask them what their strategies were for obtaining that score. Did they try any other strategies? Students should at least notice that they cannot kill wolves whenever they want. In order to kill the most wolves, the students must leave enough wolves to repopulate. This brings up the idea of *sustainability*.

2 Discuss ethical and philosophical aspects of this activity. These may include:

a) Is hunting good or bad? Neither?
b) In the activity there was a conflict of interest (you gained a better score from killing more wolves), as is the case in real life (sport value, fur coat profits, etc.). Is it fair to allow killing of as many wolves as possible, as long as the population is sustainable? Or should killing be limited to only as many as are needed to prevent overpopulation?
c) What is the niche of humans in the ecosystem?
d) Is it the human's "job" to maintain the population sustainability when it would otherwise die out? Or is it only the human's "job" to correct changes in the population sustainability that occurred due to human activities (overpopulation of a prey population due to overhunting of a predator, changes in physiology due to human effects on the environment, etc.)?
e) What does it mean for the ecosystem to be in its "natural" state? Could you argue that humans and human activities are part of the "natural" state of an ecosystem?

Student Worksheet:

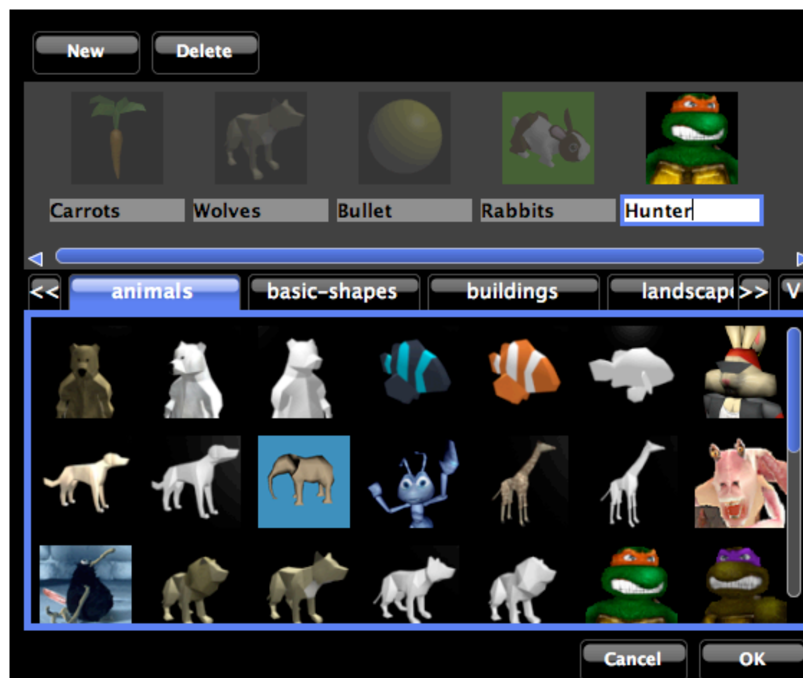
LESSON 4: HUMAN INTERACTIONS

Part 1: Program a Hunter

1) Adding a hunter breed.

    a) Click on the button next to the Factory that says **"Edit Breeds."**

    b) Click on "New" to create a new breed and give it a name (i.e. "Hunter"). Choose a shape for the breed, then click "OK."



    c) Locate the **"Setup"** page and find the **"setup"** stack.

    d) Find the **"create Hunter [num, do]"** block from the **"Hunter"** drawer under **My Blocks**.

    e) Drag it onto the **"Setup"** page and hook it under **"set Rabbits Killed"** in the **"setup"** stack. Change the number **"10"** to read **"1."**

2) Setting camera views.

If you are the hunter, you want to see Spaceland from the hunter's point of view. To do this, you must associate the first person agent camera with the hunter upon setup.

a) Click the **"setup"** button in the Runtime window and find your hunter in the middle of Spaceland. Change the point of view to **"Agent Eye"** or **"Agent View."** Notice that the screen is black because we haven't set up the camera yet.

b) Find the **"set agent camera"** block from the **"Controls"** drawer under the **Factory**. This block associates the agent camera with a given agent.

c) Drag it onto the **"Setup"** page and hook it under the **"do"** hook in the **"create Hunter"** block.

d) Find the **"ID"** block from the **"Traits"** drawer under the **Factory**. This block is the ID number of the agent (the hunter in this case).

e) Drag it onto the **"Setup"** page and stick it in the socket of the **"set agent camera"** block. Now the hunter will set the agent camera to its point of view.

f) Find the **"agent eye"** block from the **"Controls"** drawer under the **Factory**. This block ensures that we are in the **"Agent Eye"** mode every time we setup.

g) Drag it onto the **"Setup"** page and hook it under **"set agent camera"** in the **"create Hunter"** block.

3) Adding keyboard controls.

In order to control the hunter, we want to add keyboard controls. The program knows when each key is being pressed, so we only need to check that a key is pressed and perform an action if it is.

a) Find the **"Procedure"** block from the **"Procedure"** drawer in the **Factory**.

b) Drag it onto the **"Hunter"** page and rename it **"Keyboard Controls."**

c) Find the **“if [test, then]”** block from the **“Logic”** drawer in the **Factory**.

d) Drag it onto the **“Hunter”** page under **“Keyboard Controls**.**”**

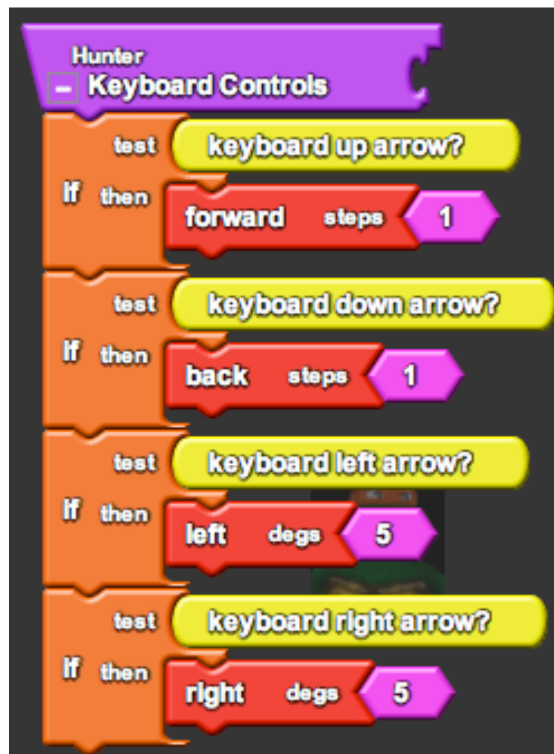
e) Find the **“keyboard up arrow?”** block from the **“Controls”** drawer under the **Factory**. This block returns true if the up arrow is being pressed.

f) Drag it onto the **“Hunter”** page and stick it in the **“test”** socket of the **“if”** block.

g) Find the **“forward”** block from the **“Movement”** drawer under the **Factory**.

h) Drag it onto the **“Hunter”** page and hook it under the **“then”** hook in the **“if”** block. Now your “keyboard controls” procedure is defined as a simple forward movement whenever the up arrow is pressed.

i) Repeat the process for **“keyboard down arrow?”** and **“back**,**” “keyboard left arrow?”** and **“left**,**”** and **“keyboard right arrow?”** and **“right**.**”** Change the numbers **“90”** to read **“5.”** Now the procedure defines movement in all directions.



j) Locate the **“Runtime”** page and find the **“forever”** block.

k) Find the **“Keyboard Controls”** block from the **“Hunter”** drawer under **My Blocks**.

l) Drag it onto the **"Runtime"** page and hook it under the **"Hunter"** hook in the **"run"** block. Now the hunter will move according to keyboard controls once you press **"run**."

4) Shooting.

For simplicity, this program already has the behavior of bullets programmed in. Any bullet that is shot in the project will travel in roughly a straight direction until either it hits a wolf or rabbit or it hits the wall. However, the hunter must shoot the bullet first.

a) Find the **"Procedure"** block from the **"Procedure"** drawer in the **Factory**.

b) Drag it onto the **"Hunter"** page and rename it **"Shoot**."

c) Find the **"hatch [do]"** block from the **"Logic"** drawer in the **Factory**.

d) Drag it onto the **"Hunter"** page under **"Shoot**."

e) Since the hunter will hatch another hunter and not a bullet, we must set the breed and appearance of the offspring. Find the **"set breed**," **"set color**," and **"set size"** blocks from the **"Traits"** drawer.

f) Drag them onto the **"Hunter"** page and hook them sequentially under the **"do"** hook in the **"hatch"** block.

g) Attach the following values for each:
   i) **"breed: Bullet"** (from the **"Bullet"** drawer in **My Blocks**) in the socket of **"set breed**."
   ii) **"black"** (open the pull-down menu under the color **"red"** by pressing the down arrow that appears on the red block) in the socket of **"set color**."
   iii) **"0.2"** (change the number **"1"** to read **"0.2"**) in the socket of **"set size**."

Now the procedure is defined as hatching a bullet (which will then travel and hit animals automatically).

h) Locate the **“Hunter”** page and find the **“Keyboard Controls”** procedure.

i) Create another keyboard control for **“keyboard space bar?”**

j) Find the **“Shoot”** block from the **“Hunter”** drawer under **My Blocks**.

k) Drag it onto the **“Hunter”** page and hook it under the **“then”** hook in the **“if”** block. Now the hunter will shoot a bullet whenever you press the space bar.

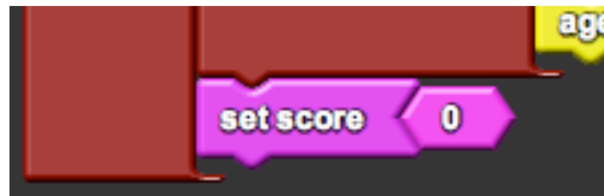
5) Keeping score.

Finally, to make a real game we must keep score.

a) Relocate the **“Setup”** page and find the **“setup”** stack from Part 1 (2g).

b) We want to start with a score of zero. Find the **“set score”** block from the **“Setup and Run”** drawer under the **Factory**.

c) Drag it onto the **“Setup”** page and hook it under **“create Hunter”** in the **“setup**” stack.

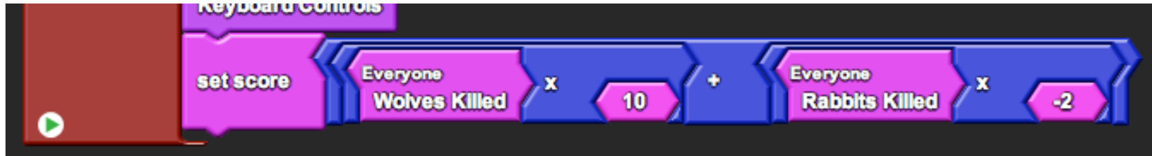


d) Relocate the **“Runtime”** page and find the **“run”** block from Part 1 (3l).

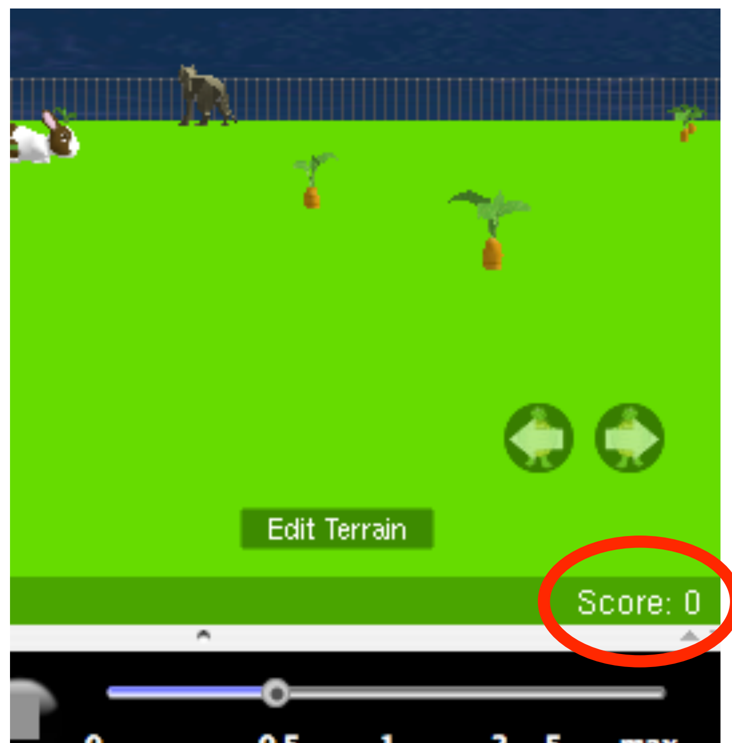
e) During each round, the hunter will keep track of the score. The score is always 10 points per wolf killed and -2 points per rabbit killed. Find the **“set score”** block from the **“Setup and Run”** drawer under the **Factory**.

f) Drag it onto the **“Runtime”** page and hook it under **“Keyboard Controls”** in the **“Hunter”** hook of the **“run”** block.

g) Attach **“(Wolves Killed x 10) + (Rabbits Killed x -2)”** in the socket of the **“set score”** block. (You can find **“Wolves Killed”** and **“Rabbits Killed”** from the **“Everyone”** drawer under **My Blocks**.)



Now you can see the score in the bottom right hand corner of Spaceland.



Part 2: Playing the game

Now that you have programmed a first-person hunter character into your ecosystem, you can start playing the game. Remember, you control the hunter by up, down, left, right arrows, and you shoot by pressing space bar.

Try to get the highest score in the class by shooting as many wolves as possible over the course of the 120 second duration of the game (it will stop running on its own when this is over). No cheating by changing the scoring rules! Try different strategies!

While you play the game, fill in the chart below. This chart will keep track of you game stats as well as the strategy you used each time you played. Also make other observations, such as if your populations die out (note the times if this happens), the highest and lowest population sizes reached, etc. Notice that you can take notes from your graphs or save the graphs themselves.

Round	Score	# Rabbits Killed	# Wolves Killed	Strategy	Observations
1					
2					
3					
4					
5					

LESSON 5: NATURAL SELECTION AND GENETIC DRIFT

Goals:
This lesson takes a step away from programming and returns to using simulation as a teaching tool. Though the students touched on the importance of chance events in the previous lesson when talking about small populations, this activity should bring the role of random events in population dynamics into focus. In particular, it is a common misconception that evolution is driven solely by this idea of "survival of the fittest." After this activity students should realize that evolution can also arise from pure chance as well as competition for resources.

Biology Concepts:
- Genetic variation (genes and alleles)
- Competition
- Genetic drift (random processes as a force of evolution)
- Natural selection (competitive advantage as a force of evolution)
- Directional, stabilizing, and disruptive selection
- Population bottleneck (chance reduction of gene pool)

StarLogoTNG Programming Concepts:
- Coordinates
- Run once

Materials:
- Starter code: *5-evo.sltng*
- Student worksheet
- Projector/ computer for demo
- Blackboard/whiteboard/large paper for brainstorming

About the Model:
The starter code contains the same rabbit and carrot ecosystem from Lesson 3. However, the rabbits in this version have an added feature: a color gene. This gene ranges from 1-100 and dictates the shade of blue that the rabbit will appear. Through the procedure, students will link the color gene to other aspects of the rabbits' survival, and this will cause there to be natural selection in the face of competition. Additionally, they will add a "flood" button that wipes out the populations on the left side of Spaceland to create a population bottleneck.

SETUP:
The model begins with 100 "carrots" and 40 "rabbits" scattered around Spaceland. Carrots begin with a random age (1-10) and random energy (1-20), while rabbits begin with a random age (1-10), random energy (1-50), and random colorgene (1-100). There is also an "observer" who just keeps track of the colorgene values for data purposes.

FLOOD:
The flood button causes all agents on the left side of Spaceland ($x<0$) to die.

RUN (120 sec):
Carrot procedures:

- Increase age +1
- Update visual height
- Produce energy +2 (if fewer than 10 carrots in radius 10)
- Reproduce (if energy > 20)
- Die (if age > 50)

Rabbit procedures:

- Turn toward any nearby carrots if possible and move forward [0.15*colorgene + 0.25] (average 1 over all colorgenes) steps.
- Increase age +1
- Decrease energy [-0.015*colorgene] (average -0.75 over all colorgenes)
- Reproduce (if age > 15 and energy > 15)
- Die (if age > 40 or no energy)
- Eat carrots during collisions (gain 50% of the carrot's energy)

Observer procedures:

- Sums the colorgene values for every rabbit.
- Calculates the average colorgene value and the max/min/range of the colorgene.

GRAPH:
The first graph shows the current carrot population divided in half (line 1) and the current rabbit population (line 2), as in Lesson 3. The second graph shows the average colorgene value (line 1) and the range of colorgene values (max – min, line 2). The bar graph shows the distribution of colorgene values (bar 1 = rabbits with color 100, bar 2 = rabbits with color 101, …, bar 10 = rabbits with color 110).

Possible Modifications:
The colorgene can be linked to many different aspects of the rabbits' survival, so you may choose to link it to something else. Also, if time is limited, you may choose to remove the population bottleneck aspect of the activity, or preprogram the flood in the starter code.

Suggested Lesson Guide:

Part 1: Discussion

1 What is evolution? How does it come about?
   a) Brainstorm a list on the board.
      - Use this to gauge initial student understanding about evolution and the misconceptions that may need to be cleared up.
      - You can also refer back to this list during the next lesson on mutation.
   b) Highlight two different forces: genetic drift and natural selection. Note that these are the forces that will be emphasized in the activity.

2 Discuss the idea of alleles and genetic variation. What role does it play in evolution?

Part 2: Activity

1 Begin worksheet activity.
   a) Assign driver/navigator pairs, get the students on the computers, and open the file.
   b) Note that the model is the same bunny model that the students programmed with one main difference. Ask the students to click setup and see if they notice that the rabbits are now colored. Explain that the color gene only changes the color and has no effect on rabbit survival.

2 Color gene correlated to color only.
   a) Ask the students to hypothesize the outcome.
   b) Allow the students to experiment with the model and fill out the chart provided.
   c) Discuss the results. The surviving colors should be different for most students between each run.
   d) Highlight that the chance events caused certain genes to be selected in each round, although there was no selective pressure dictating which genes those were.

3 Color gene correlated to speed.
   a) Demo the change in rabbit speed in the code and ask students to do the same.
   b) Ask the students to hypothesize the outcome.
   c) Allow the students to experiment with the model and fill out the chart provided. You can probably cut this experiment to two or three runs, since one color almost consistently always survives.
   d) Discuss the results. The surviving colors should be the fastest rabbits in each run, causing *directional selection*.
   e) Highlight the competitive advantage that the fast rabbits have in this scenario.

4 Color gene correlated to speed and energy loss.
   a) Demo the change in rabbit energy loss and ask students to do the same.
   b) Ask the students to hypothesize the outcome.
   c) Allow the students to experiment with the model and fill out the chart provided.

d) Discuss the results. This experiment is not so clear-cut. Most times there will be a *stabilizing selection*, but occasionally there will be *disruptive* selection, and few times there will be seemingly random behavior. Hopefully the large quantity of experiments that the class performed collectively may show a certain trend.
e) Ask the students to hypothesize when stabilizing selection might occur and when disruptive selection might occur in the same system. (The following are also guesses – they haven't been tested yet.)
    - One hypothesis for why stabilizing selection occurs most of the time because fast rabbits will lose energy too quickly and slow rabbits cannot compete for resources.
    - One hypothesis for when directional selection might occur is that slow rabbits that are somewhat segregated in a corner at the start of the simulation do not have much competition from faster rabbits and therefore have time to populate to large numbers due to their slow energy loss. Then when they begin to mix with faster rabbits they survive by sheer number. This hypothesis may be tested out by physically segregating the rabbits to begin with.

5 Population bottleneck.
a) Demo the programming portion of creating a "flood" which kills all agents on the left side of Spaceland and ask students to do the same.
b) Let the students play with the model using the flood button at strategic times. Let them choose what their goal is. They may try to wipe out all of the rabbits of one color, or try to do as little damage as possible, or even just press randomly to see what happens.
c) Ask the students to share how their populations evolved when they created floods. Highlight the idea of *population bottleneck* as an event where a certain portion of the gene pool survives by chance.

Part 3: Debrief

1 Ask the students what they learned about evolution from the activity.

2 Discuss the model.
a) What other forces can drive selection? What other natural events can create population bottlenecks?
b) Does more variety happen when there is selection or when there is not?
c) Is there anything missing from the model?

Student Worksheet:

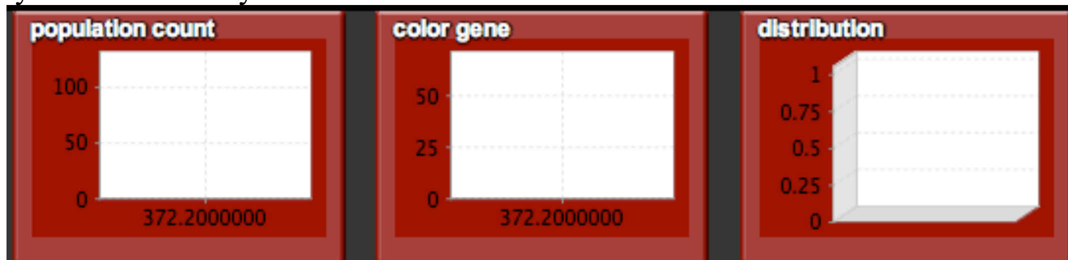
LESSON 5: NATURAL SELECTION AND GENETIC DRIFT

Part 1: The color gene

The rabbits have been given a "color gene" that ranges from 1-100 and determines what shade of blue they appear.

Notice the three graphs in the Runtime window.

- The first is the usual graph of carrot and rabbit population sizes.
- The second is a graph of the average value of the rabbit color gene and the range (difference between the highest and lowest) of the color gene.
- The third is a bar graph showing the distribution of the color gene. This graph will let you see how many rabbits are of each color.



Run the simulation a couple of times. You may run each trial until either the simulation stops or all but one color of rabbit survives. As you run each trial, fill out the first chart at the end of the worksheet.

Part 2: Rabbit speed

Link the color gene to the speed of the rabbit. Rabbits with higher color gene values run faster. We will associate the two traits by the equation:

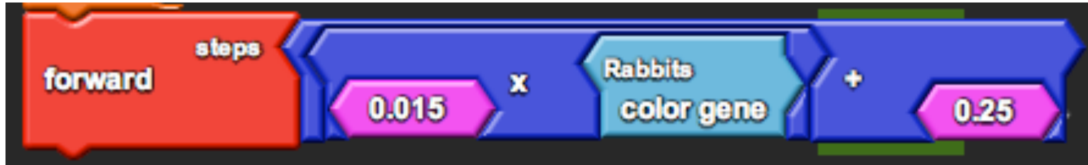
rabbit speed = (0.015 x color gene) + 0.25

1) Locate the **"Rabbits"** page and find the **"Hop"** procedure.

2) Change the value **"1"** in the socket of the **"forward"** block to say:

"0.015 x color gene + 0.25"

You can find the **"color gene"** block from the **"Rabbits"** drawer under **My Blocks**.



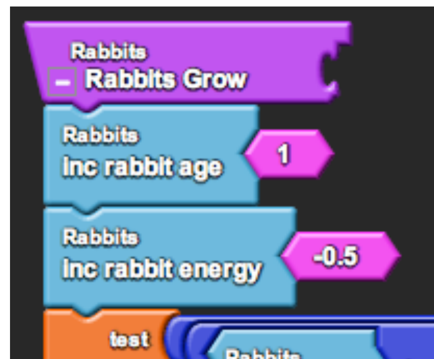
Run the simulation a couple of times. You may run each trial until either the simulation stops or all but one color of rabbit survives. As you run each trial, fill out the second chart at the end of the worksheet.

Part 3: Rabbit energy loss

Link the color gene to energy lost by the rabbit (presumably rabbits that run faster need to burn more energy). Rabbits with higher color gene values lose more energy. We will associate the two traits by the equation:

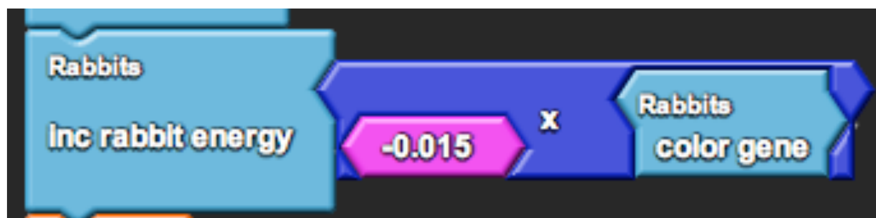
rabbit energy gain = -0.015 x color gene

1) Locate the **"Rabbits"** page and find the **"Rabbits Grow"** procedure.



2) Change the value **"-0.5"** in the socket of the **"inc rabbit energy"** block to say:

"-0.015 x color gene"



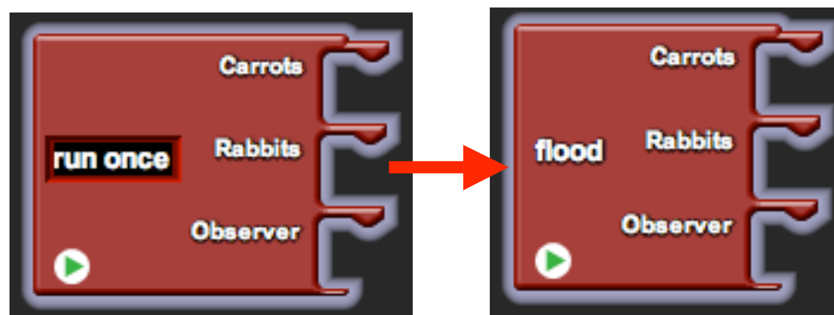
Run the simulation a couple of times. You may run each trial until either the simulation stops or all but one color of rabbit survives. As you run each trial, fill out the third chart at the end of the worksheet.

Part 4: Flood!

Populations normally suffer from the effects of chance natural events. We will program a "flood" that wipes out the carrot and rabbit populations on the left side of Spaceland. (Can you think of other natural disasters that may occur?)

1) Find the **"run once"** block from the **"Setup and Run"** drawer under the **Factory**. This block runs the given procedures only one time (unlike the **"forever"** block, which loops over the procedures forever, and the **"run"** block, which runs the procedure for a given amount of time).

2) Drag it onto the **"Runtime"** page and rename it **"flood**."



3) Drag an **"if [test, then]"** block under each of the **"Carrots"** and **"Rabbits"** hooks.

4) Find the **"xcor"** block from the **"Traits"** drawer under the **Factory**. Spaceland is organized as a giant grid with an x-y coordinate system. This block returns the x-coordinate of the agent.

5) Agents on the left side of Spaceland have a negative x-coordinate. Attach **"xcor < 0"** in the **"test"** socket of both **"if"** blocks.

6) Finally, we want those agents to die when there is a flood. Attach **“die”** in the **“then”** socket of both **“if”** blocks. Now as you run the simulation using the **“run”** button, you can also press the **“flood”** button to flood the left side of Spaceland.

Run the simulation a couple of times. You may run each trial until either the simulation stops or all but one color of rabbit survives. As you run each trial, fill out the final chart at the end of the worksheet.

During the different trials you may choose to administer the flood at strategic times to kill of certain populations. You may also choose to administer the flood totally randomly. Try different strategies and note the outcome of each. Remember that floods do not choose when to occur, but using a simulation we can test the different possible ways in which they can.

Color gene (unlinked):

Round	Stop Time	Average Color Gene	Range Color Gene	Rabbit Colors Surviving (and number of each)	Observations
1					
2					
3					
4					
5					

Color gene linked to speed:

Round	Stop Time	Average Color Gene	Range Color Gene	Rabbit Colors Surviving (and number of each)	Observations
1					
2					
3					
4					
5					

Color gene linked to speed and energy loss:

Round	Stop Time	Average Color Gene	Range Color Gene	Rabbit Colors Surviving (and number of each)	Observations
1					
2					
3					
4					
5					

Color gene during flood:

Round	Stop Time	Average Color Gene	Range Color Gene	Rabbit Colors Surviving (and number of each)	Observations
1					
2					
3					
4					
5					

LESSON 6: MUTATIONS

Goals:
This lesson builds on the previous lesson about evolution. The goal of this lesson is for students to learn where the genetic variation from the previous lesson comes from and why it is important. In particular, students should understand that the idea of an individual's "fitness" is an idea that exists solely in the context of a given environment. When the environment changes, genetic variations within the population may allow the species to adapt and survive, though the individuals may die.

Biology Concepts:
- Mutations
- Adaptation
- "Fitness" as a term defined by the current environmental conditions

StarLogoTNG Programming Concepts:

Materials:
- Starter code: *5-mutation.sltng*
- Student worksheet
- Blackboard/whiteboard/large paper for brainstorming

About the Model:
The code contains the same rabbit and carrot ecosystem from Lesson 5. However, there is an added element in this model of temperature. Although the carrots from Lesson 1 grew better or worse at different temperatures, we ignore the effects of temperature on the carrots in this case and focus on the effects of temperature on the rabbits. The carrots affect the rabbit energy loss (explained below).

MODIFICATIONS IN THE CODE (FROM LESSON 5):

The key modification is that each breed (carrots and rabbits) has a list variable called "DNA." The DNA holds a list of values for "genes." Note that the genes are numerical in this case, because the mutation code increases or decreases the gene value by 20%. All agents of a breed begin with the same exact genetic makeup. The initial values are set in "set avg <breed> genes," and each agent is assigned these initial values in "set DNA." Like in Lesson 5, there is an observer who calculates the average gene values in each round, but the code is slightly modified to deal a list of values rather than a single value.

Though the setup allows for many genes, there is only one gene currently programmed: the color gene from the previous activity. However, instead of creating a separate number variable called "color gene," we add the initial gene value as "50" in "set avg rabbit genes." Now, whenever we want to reference the color gene, we reference "get list item [list, index]" from list "DNA" at index "0" (note that all lists are indexed from 0 (i.e. first item is index 0, second item is index 1, etc.)

There is no change in the speed of the rabbit from Lesson 5, but the energy lost by each rabbit depends on both color gene *and* temperature in this activity. At high temperatures the energy loss has a relatively linear dependence on color gene value (with low values having low energy loss and high values having high energy loss), but at low temperature the energy loss is more exponential. In this way, the difference in energy loss between gene values is greatly amplified at low temperatures. The actual equation used is:

$$\text{energy loss} = 2 \wedge (\text{color gene} / (\text{temperature}*0.6 + 40)) - 1$$

The specific parameters are such that rabbits with color gene 50 survive at temperature 50 degrees but die out at temperature 0. The behavior follows the following graph:

Energy Loss

Temperature

Gene value

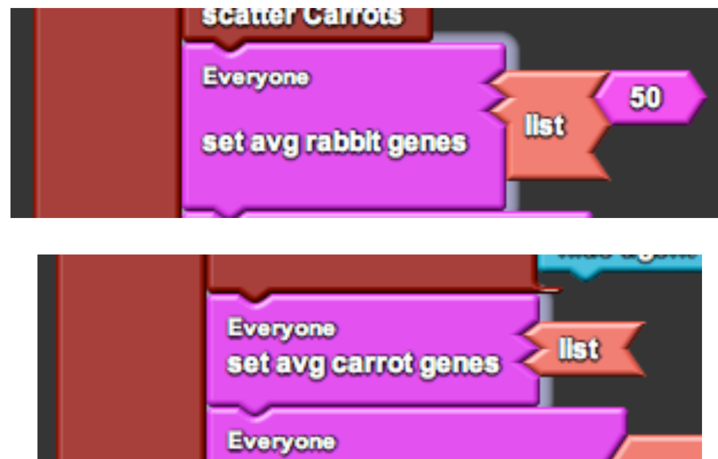
GRAPH:
The first graph shows the current carrot population divided in half (line 1) and the current rabbit population (line 2), as in Lesson 3. The second graph shows the mutation rate (line 1), temperature (line 2), and average colorgene value (line 3). The bar graph shows the distribution of colorgene values, with the first and last bar counting rabbits with colors outside of the range of blue (bar 1 = rabbits with color < 100, bar 2 = rabbits with color 100, bar 3 = rabbits with color 101, …, bar 11 = rabbits with color 110, bar 12 = rabbits with color > 110).

Possible Modifications:

The code supports the easy addition of more numerical genes in the DNA of both rabbits and carrots. The "observer" agent performs all of the calculation related to it, and the "Mutate" procedure performs the actual mutations (you may change the mutation behavior through this procedure).

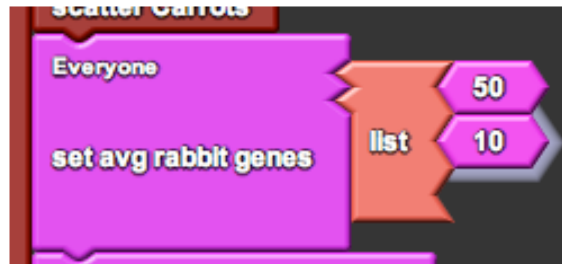
To add a gene:

1) Locate the **"Setup"** page and find the **"setup"** block. In this block there are **"set avg carrot genes"** and **"set avg rabbit genes"** blocks.



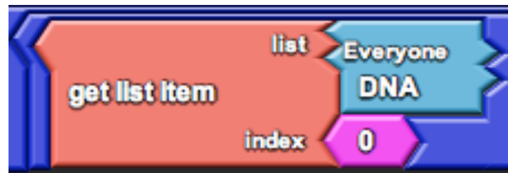
2) Notice that the carrot genes is an empty list (there are no carrot genes to keep track of) and the rabbit genes is a list with one entry of 50 (there is one rabbit gene that initiates with value of 50).

3) Add a gene by adding a value to the list. For example, to add a second rabbit gene (let's refer to it as "RG2") that initiates with value 10:



To use the added gene:

Get the gene with the "get list item [list, index]" with [list] being the DNA and index being the number of the gene (with number beginning at zero). For example, the first gene, the "color gene," is indexed at 0, so a rabbit can access the color gene with:

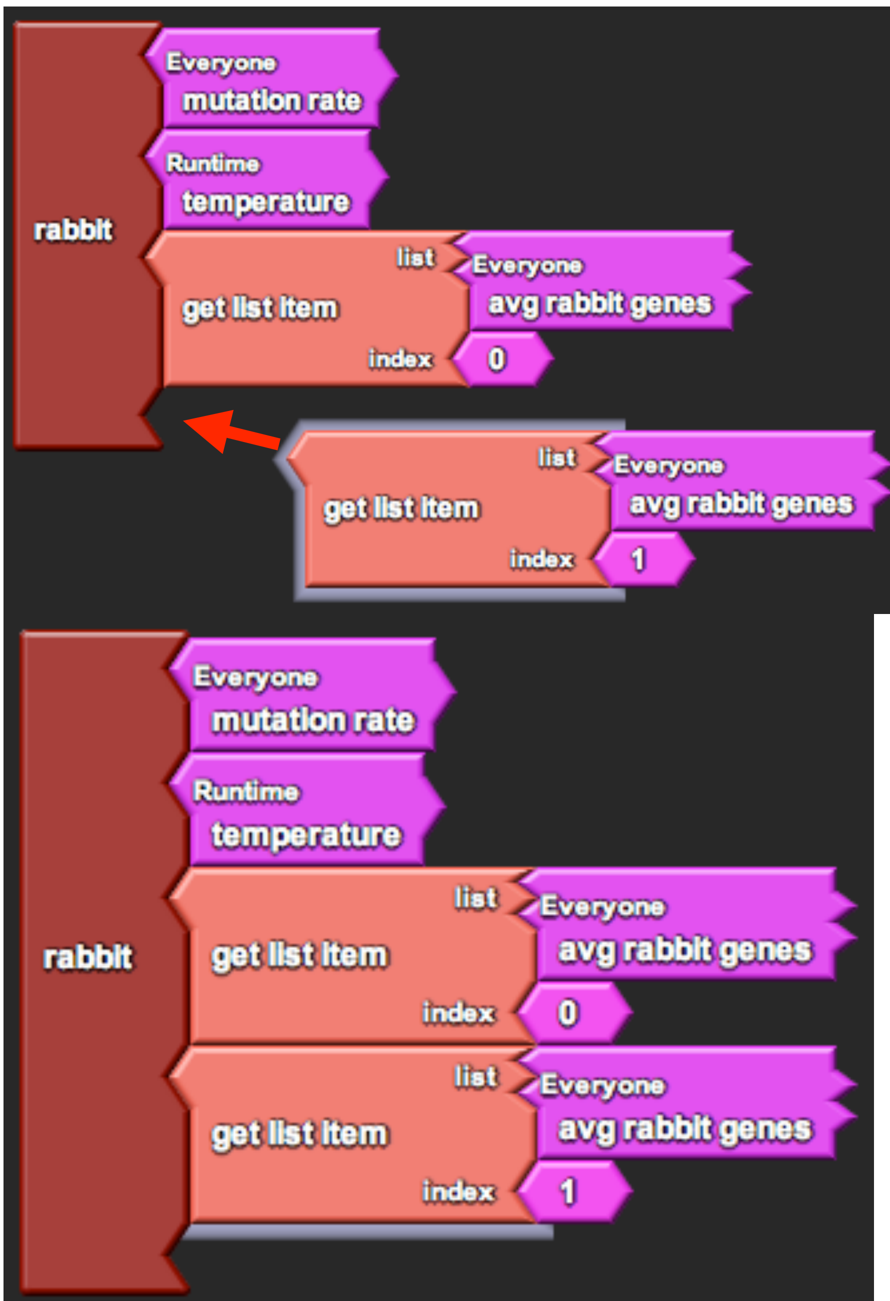


This returns the numerical value of the color gene. If we wanted to access the second gene, we would get the list item at index 1.

To graph the average value of the added gene:

1) Locate the **"Runtime"** page and find the **"rabbit"** block. This block graphs values related to the rabbit. (There is a similar **"carrot"** block. To use it, replicate the code in the **"rabbit"** block.)

2) Add a gene by getting the gene and adding it to the graph. For example, to add the gene "RG2":



rabbit
Everyone
mutation rate
Runtime
temperature
get list item
list
Everyone
avg rabbit genes
index
0
get list item
list
Everyone
avg rabbit genes
index
1
rabbit
Everyone
mutation rate
Runtime
temperature
get list item
list
Everyone
avg rabbit genes
index
0
get list item
list
Everyone
avg rabbit genes
index
1

Suggested Lesson Guide:

Part 1: Discussion

1 Are mutations good or bad?
   a) Poll the students to see if they think mutations are good or bad.
      - Students may understand mutations as defects that arise, creating things like cancer or strange deformities.
   b) Note that mutations are a source of genetic diversity. Recall the color gene in Lesson 5 and note that, though there was no mutation in the simulation, the different color gene values must have initially arisen from mutation.

2 Discuss the mechanics for how mutations occur.
   a) Note that there are varying degrees of mutation (silent to deadly).
   b) Note that there are different types of inheritance ("yes or no" type genes, or "how much" type genes). In this model we use "how much" type genes.

Part 2: Activity

1 Begin worksheet activity
   a) Assign driver/navigator pairs, get the students on the computers, and open the file.
   b) Note that there is an added element in the system: temperature.
      - Students may recall that the temperature affected carrot growth in Lesson 1, but make sure to explain that it only affects the rabbits in this simulation
      - Students may also recall that there was genetic variation amongst the rabbit color genes in Lesson 5, but make sure to note that this system begins with all rabbits having the same color gene (variation will only arise from mutation).
   c) Ask the students how well they think the rabbits will survive at different temperatures. Ask them to hypothesize how mutations will affect this survival.
      - Make sure to note that the mutations occur during reproduction, so they can only affect survival of the species and not of individual rabbits.

2 Varying the sliders
   a) When students vary mutation rate, they should notice a rise in the number of different colors of rabbits, which correlates to an increase in variation.
   b) When students vary temperature (mutation rate 0), they should notice that the rabbit population survives in higher temperatures but dies out in lower ones.
   c) When students vary temperature with high mutation rate, they should notice that the average value of the color gene increase slightly at higher temperatures and decrease greatly at lower temperatures. Additionally, a more gradual change in temperature may allow more time for genetic variation to build up.
   d) For the final open-ended investigation, remind students to record their observations. An activity they can try is to vary mutation rate first to create variation, then set it back to 0. The initial variation may be enough to now survive temperature changes.

Part 3: Debrief

1) Ask students to present their strategies for varying the parameters and how these different cases affected the rabbit population and average gene value.
   a) Ask them to support their conclusions based on the data that they collected.
   b) See if they could predict what the actual model behavior was (how the rabbit growth was affected by temperature). They will likely not be able to guess the exact behavior, but they may understand that the discrepancy between energy loss of low and high color gene values is amplified at low temperatures.

2) What is evolutionary fitness?
   a) Emphasize that the ability of an allele to survive in a population depends on the current environment. What is considered good in one environment might be bad in another.
   b) Note that mutations are important in cushioning the population in the case of a change in the environment.

3) Discuss the model.
   a) What other genes can be added to the model? How would students implement these genes based on what they know?
   b) What is missing from the model? (ex. Sexual reproduction)
   c) Note that the model allowed us to test a phenomenon that may take many (many) years to observe in real life.

Student Worksheet:

LESSON 6: MUTATIONS

In this activity there are two sliders that you can control: ***mutation rate*** and ***temperature***. Record your observations on a separate sheet of paper or on the chart provided at the end of the worksheet. Make sure to note the temperature and mutation rate of each experiment, and whether you changed the values during setup or while the simulation was running. You may also choose to save your graphs (use the **"Save Image"** button on the graph).

Part 1: Mutation rate only

Set the temperature to the default temperature of 50 degrees.

1) Try setting up with a mutation rate of 0 and note how the average value of the color gene evolves over time.

2) Repeat the procedure with different mutation rates 50 and 100.

Part 2: Temperature only

Set the mutation rate to the default value of 0.

1) Try setting up with temperatures 0 and 100 (instead of 50, as in Part 1).

2) Now try setting up with temperature 50 and decrease the temperature to 0. Repeat the procedure, instead increasing the temperature to 100.

Now set the mutation rate to the highest value of 100.

1) Repeat the procedure with setup temperatures 0 and 100.

2) Repeat the procedure with gradually decreasing/increasing temperatures to 0 and 100. Try varying the speed with which you decrease and increase the temperatures.

Part 3: Mutations and Temperature

Conduct your own experiments while varying both mutation rate and temperature. Try changing the values at different times in different orders. Try decreasing and increasing values within the same experiment.

How does the speed and timing of your variations affect the evolution of the system?

Round	Temperature	Mutation Rate	Average Color Gene	Notes/ Observations