Playing with Parameters

Part 1: Prepare to build the project

1. To model a system, you first need to discover some things about the system. One of the most common types of ecological models is a predator-prey model. This is a model of a system of two animals, where the predator animals eat the prey animals. The basic questions you should ask when making a predator-prey model are listed below:

    a. Which species is the predator?
    b. Which species is the prey?
    c. What does the prey eat?
    d. How much food does each species need to survive?

2. For the dinosaur ecosystem, we will simplify things a little bit. Even though this will make the simulation less realistic, it will make it easier for you to learn the basics of programming models and simulations. Here is the information about the basic dinosaur ecosystem model that you will build for this project:

    a. Triceratops eats green leafy bushes.
    b. T-Rex is the predator that eats Triceratops as its prey.
    c. All dinosaurs spawn a new dinosaur after consuming enough energy from food.
    d. When one of the species of dinosaurs runs out of food, the simulation ends.

3. Open Tynker on your device.

4. Create a new blank project.

5. Delete the blue default actor.

6. Rename the project to “P4 DinoEco” plus your initials; for example, “P4 DinoEco AB”.

7. Add the T-Rex actor from the Adventure category.

8. Add the Triceratops actor from the Adventure category.

9. Add a bush of your choice from the Adventure category.

10. Resize and rename the actors in the Properties window.

    a. t-rex, scale = 20
    b. triceratops, scale = 15
    c. bush, scale = 100

11. Give your stage a background scene. (stage → properties → add scene)

12. Set your Tynker project aside and continue to Part Two to explore the problem in more detail.

Part 2: Use the steps of the Design Process to create a dinosaur ecosystem simulation

Like other scientists, computer scientists use a **design process** to create **solutions**. The design process has five major steps. You will follow these steps as you work on your project.

- Ask: What is the problem?
- Explore: What are ways to solve the problem?
- Model: Create a solution for the problem.
- Evaluate: Test to see if the solution works.
- Explain: Talk about how and why your solution works.

13. **Ask**

    Think about answers to the following questions. Or, if you've printed these directions, write the answers next to the questions.

    a. What is the problem that needs to be solved?
    b. What are the agents in this system?
    c. What are the parameters?

14. **Explore**

    This is the step where you explore options for building your ecosystem model and simulation. Brainstorm and write your ideas down.

    d. What are the requirements? Break the requirements down into small statements. Answer each question below.

- When the program starts, what initial values do you need to know? Think about the parameters of the simulation.
- What **events** happen in the simulation?

e. Sketch the basic parts of the program using pictures, diagrams, and/or pseudocode. A launch log page may be downloaded and used. *See the blog.
Use the ideas from your notes from the step above and look at the list below for ideas that you may not have considered. For now, just write them down. You will begin programming in the next step.

- Make **variables** to hold initial values and to track values as they change during the simulation.
- Create the correct number of t-rexes, triceratops, and bushes based on the values specified in the parameters.
- What happens when a t-rex and a triceratops touch each other? The t-rex gains energy and the triceratops dies.
- What happens when a triceratops touches a bush? The triceratops gains energy and the bush disappears.
- When a dinosaur's energy is equal to the energy required to spawn, create a new clone and reset the dinosaur's energy to 0.
- Check the number of food sources (triceratops and bushes). If either is equal to zero, end the simulation and display the time and final number of each actor.

Pseudocode	Tynker code
Repeat continuously: *Pick a random direction* *Move in that direction* *Bounce if hitting edge of screen*	

Pseudocode is similar to computer code but is easier for humans to understand. Programmers use it to sketch out an algorithm before starting to write a program on a computer.

15. **Model**

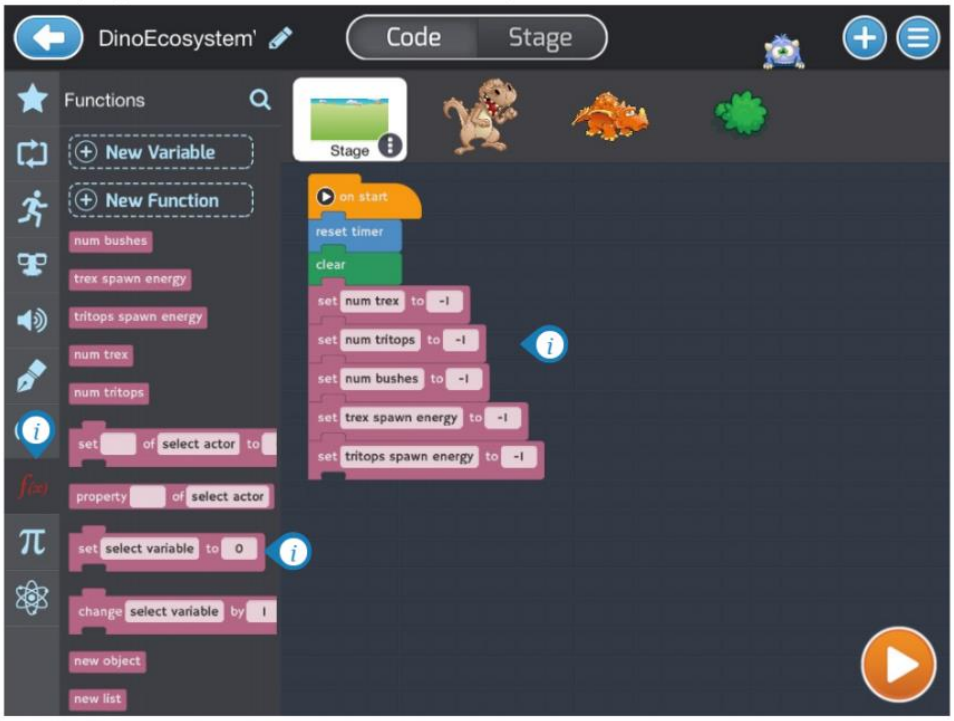
a. This is the step where you start building your program. The steps below will help you to break the program down and will give you information you need to know to successfully build the simulation.

b. Make variables to hold initial values of the parameters and to track values as they change during the simulation. Choose short, meaningful names for the variables.

- Refer back to our last activity if you need a reminder about how to create new variables. Create the following **global variables in the stage's code area**. (Functions category, red)

- number of t-rex dinosaurs
- number of triceratops dinosaurs
- number of bushes
- t-rex spawn energy
- triceratops spawn energy

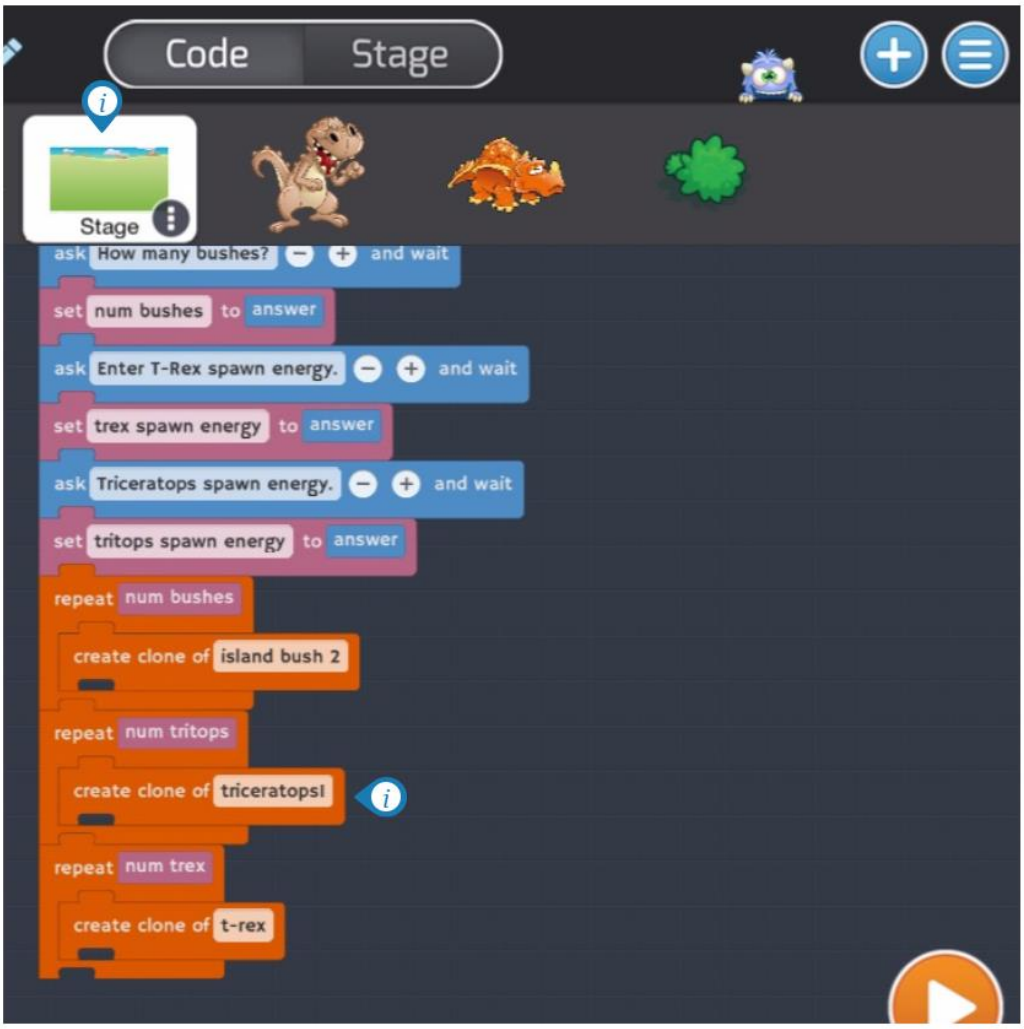
c. Variables start out as 0. The *when num bushes=0* ***or*** *num tritops = 0* event triggers when the value of one of those variables falls to 0. To prevent this we set all the variables to -1 on start. If variables started out at 0 then this event would trigger as soon as the program started and that would be a bug!



d. When the simulation starts, ask the user to enter starting values for the initial number of t-rexes, number of triceratops, number of bushes, energy for t-rexes to spawn, and energy for triceratops to spawn. Set this code up in the **stage's code area**.

> **Note:** Use the *ask and wait* block to ask for a parameter. Store the answer in the appropriate global variable. The *ask* block is in the Sensing category (light blue) and it says "What is your name" in its white box—click on the question to change it. The *answer* block is also in the Sensing category. Variable blocks are in the Function category (red).

e. Set up all of the actors: t-rex, triceratops, bushes.
   - Think back to the dodgeball tilt game when you created more than one ball by creating clones. In this project you need to make many dinosaurs and bushes. It would be difficult to make a new actor for every agent in this simulation. Cloning is essential in modeling and simulation because it allows you to program a single agent and then copy that agent as many times as necessary.
   - For now just create the clones for each agent. You will program the agents in the next step.
   - The example code that follows uses three repeat loops to set up the clones for each type of agent. Where did the variables *num bushes*, *num tritops*, and *num trex* come from? Do not proceed until you understand why this code works.

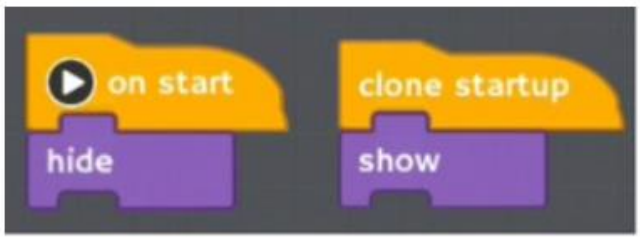


The *repeat* block and the *create clone* block are in the Control/Flow category (orange). Variables are in the Function category (red). Your names may be different than the example.

f. **Test** your code. Your program should ask the user for the initial parameters. Then what happens? Do you see any clones on the screen? No! That's expected because you have not yet programmed the *clone start* event for each actor.

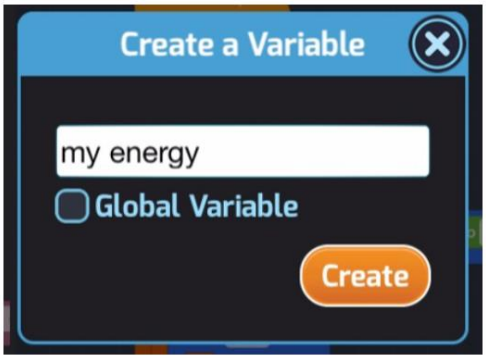
g. Program each actor's *clone startup* script.

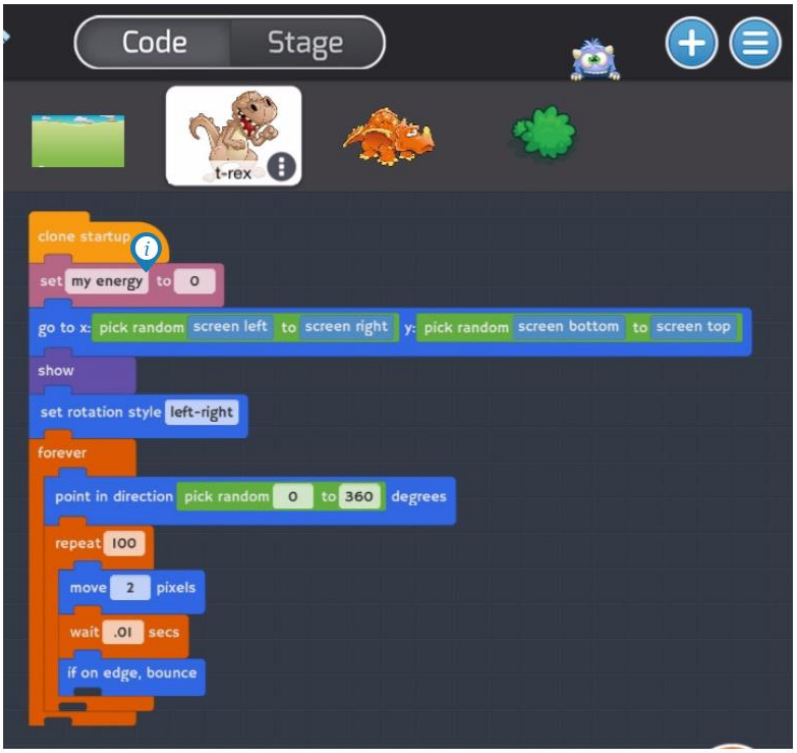
- You must *hide* the original actor and put a *show* block in the *clone startup* script. This will make it easier to create the right number of clones for each actor. The clones do not run the *on start* script. They start off hidden because they inherit the hidden property from the original actor. That is why you have to use *show* during *clone startup*.



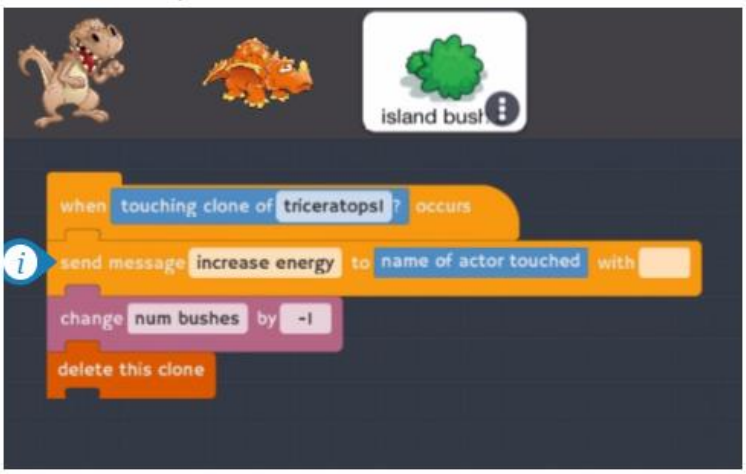
Use hide and show for all three types of clones

h. Make the t-rex and triceratops clones start at a random location on the screen and then move around randomly. Hint: Refer to the tilt game to see how you made the kid actor move randomly on the screen.
i. Position the bushes all around the screen. Remember that they should not move! Hint: Bushes only need the first part of the *clone startup* script that the two types of dinosaurs have. Refer back to the code for the treasure chest in the tilt game for more help.
j. Test your code. Your program should ask the parameter questions and then create the correct number of dinosaurs and bushes. Dinosaurs should move randomly while bushes stay still.
k. Program the dinosaurs to keep track of their own energy.
    - Set up a <u>local</u> variable for the t-rex actor. You may name the variable whatever you like. Use a name that indicates what the variable represents, such as t-rex-energy.
    - Set up a <u>local</u> variable in the triceratops actor. Name the variable something that indicates what it represents, such as tritops-energy.
    - Remember to initialize these local energy variables to 0 in the dinosaur *clone startup* code. The image to the right gives an example of how to set up an energy variable. <u>Do not check Global Variable.</u>





- In this example, the original actor has a **local variable called *my energy***. Therefore, each clone of the actor will have <u>its own *my energy*</u> variable.
    l. Program the events for actors touching each other.
    - When a triceratops touches a bush, the triceratops's energy increases by 1 and the bush disappears.
    - When a t-rex touches a triceratops, the t-rex's energy increases by 1 and the triceratops disappears.
    - Remember to update the variables that keep track of the number of each type of actor. (*num bushes* and *num tritops* in the example)
    - To get these behaviors to work correctly you will need to use *send message to actor* and *when I receive message*. These blocks allow the clones to talk to one another. See the hint in the picture below.



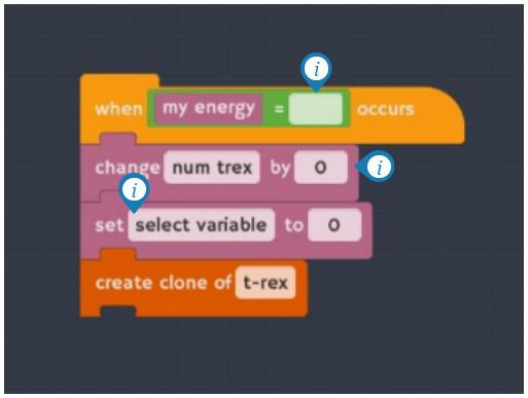
Sending and Receiving Messages

- Look closely at how the bush tells the triceratops that they have touched. Use this to help you figure out how the triceratops should tell the t-rex that they have touched. Which actor should disappear? Which actor should increase its energy?
- Remember to <u>use *touching **clone** of t-rex*</u> and not *touching t-rex*.

- In this example the message is named "increase energy," but you may use any name you like. It is OK to leave the *with* space blank on the *send message* block.
- Clones run all of the scripts in the actor that cloned them, except for the *on start* script (because they run *clone startup* instead). That means that if you put a script in t-rex's program area, all t-rex clones will run that code as well.

m. Test your code. Your program should ask the parameter questions and then create the correct number of t-rex clones, triceratops clones, and bush clones. When a t-rex bumps into a triceratops the triceratops should disappear. When a triceratops bumps into a bush, the bush should disappear.
   - Remember that it's normal to go through cycles of testing and fixing when you are writing a program. If anything is not working correctly be persistent until you have fixed the **bug**!

n. Program the **t-rex clones** to reproduce when their own *my energy* reaches the *t-rex spawn energy* parameter. A hint about the blocks you will need is provided below. Fill in the blank spaces. Remember to reset *my energy* to 0 after spawning so that the dinosaur can work its way up to spawning again.

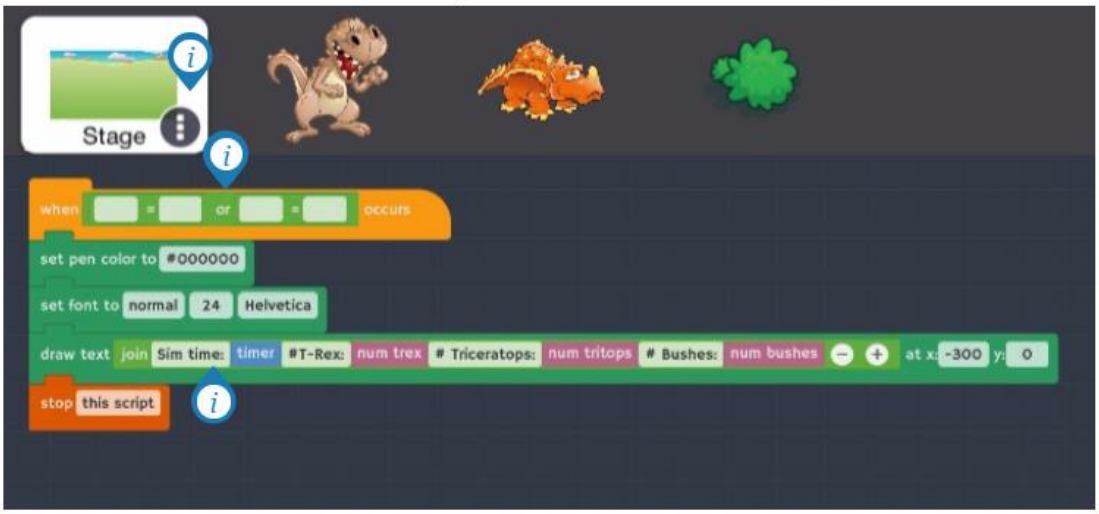


o. Program the **triceratops clones** to reproduce when their energy reaches the triceratops spawn energy parameter. The code will look very similar to the code that you just wrote to make the t-rex spawn except that you will put it in the *clone startup* script for the triceratops actor and not the t-rex actor.

p. Test your code. Your program should ask the parameter questions and create the clones. When a t-rex bumps into a triceratops the triceratops should disappear. When a triceratops bumps into a bush, the bush should disappear. New t-rex clones and new triceratops clones should be popping up every now and then whenever one of the dinosaurs has earned enough energy to spawn. Hint: Remember that the *say* block is useful for displaying the current value of a variable. You can use it to see what is happening while the program runs.

q. End the simulation when one of the food sources runs out. Add a *when true occurs* event to the **stage's script** that is triggered when the food source for either dinosaur runs out.
   - What is the condition you want to watch? What should happen when that condition becomes true? Use the hint that follows to help you program the

ending condition for the simulation. Fill in the blank spaces in the *when true occurs* block.



- Fill in the *when true* block to detect the condition of either the number of bushes or the number of triceratops reaching zero. To detect either one condition or another, use a *true or true* block.
- The *stop [all]* block when changed to *stop [this script]* will stop the text from appearing more than once. Be sure to update the data and draw the text showing the final data before stopping the program.
- *When true* and *stop* blocks are in the Control/Flow category (orange). *Draw*, *set font*, and *set pen* blocks are in the Drawing category (dark green). *Join*, *or, and* = blocks are in the Math category (light green).
- The example above shows a *timer* block in the *draw text* block. Use the *timer* variable block later in the program to find out how long the program has been running. You must initialize the timer when the program starts by putting a *reset timer* block in the stage's *on start* script. Timer blocks are in the Sensing category (light blue).
- You should also put a *clear* block in the beginning of the stage's *on start* script to clear away any text that is still there from a previous simulation.

r. Test your code. Your program should ask the parameter questions and create the clones. When a t-rex bumps into a triceratops the triceratops should disappear. When a triceratops bumps into a bush, the bush should disappear. New t-rex clones and new triceratops clones should be popping up every now and then whenever one of the dinosaurs has earned enough energy to spawn. When there are no more bushes or no more triceratops, your program should display the data from the simulation and end the program.

16. **Evaluate**
    a. Test your code very regularly as you build your program so that it is easier to isolate and fix any issues (also known as bugs) in the code. Waiting until the program is completed to test the code could make it confusing and difficult to fix any problems. Remember, finding bugs and fixing them is known as **debugging** in computer programming.
    b. It is normal and expected that programs may not work exactly as intended at first. Be patient and make sure to iterate between programming and testing as many times as needed.
    c. A good way to know whether your variables are updating correctly is to display them on the screen using the *say* block. For example, if you want to know an actor's energy, you could put "say (my energy)" inside the actor's *forever* loop.
    d. If you see that recent changes you made to your program are not showing up when you test (especially when it comes to variables), it is always good to go back to the My Projects screen and reload your project. Sometimes the Tynker app gets hung up on something and needs a fresh start to get back to working correctly. Simply

go back to the My Projects screen by clicking the blue arrow in the upper-left corner. Then go back into your project.

e. When your Dinosaur Ecosystem simulation is working according to all of the requirements, you can use any extra time you have to improve your project. Here are some ideas that you might like to implement if you have extra time.
   - Display the t-rex and triceratops numbers on the screen while the simulation is running, like you did with the game score in the dodgeball game.
   - Add sounds (to the actor's properties) and play a sound when the actors bump into each other.
   - Keep track of how long a dinosaur has been alive and make it die after a certain amount of time. Could you make this lifespan a parameter in the simulation?

17. **Explain**
    a. If possible, share your simulation with another student virtually and explain your code.
    b. Have them share their program with you as well.
    c. Discuss a few issues that you or the other group encountered and how you solved them.
    d. As part of explaining your project, you should be able to demonstrate how it can be used to understand the dynamics of the dinosaur ecosystem you have modeled. Run simulations with several different parameters and record your observations. Set up a chart like this and use it to record your simulations. Be sure to change parameters as you test, but don't change too many parameters at once. Changing one or two parameters in each simulation can help you understand the cause and effect going on inside the system.

Sim #	Parameters	Outcomes	Notes