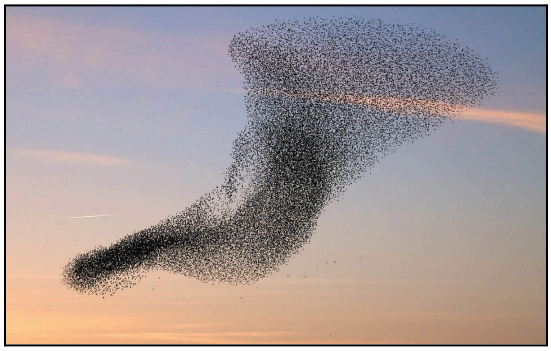
What is Complexity Theory?

Close your eyes and think of your favorite hobby. Now, assuming you've opened your eyes and can read this next sentence, grab a pen and paper to start listing all of the pieces that come together to make your favorite hobby come alive. Maybe you pictured a video game, involving storyboards, code, design, marketing, and distribution. Perhaps it was reading a novel, you need chapters, paragraphs, characters, backgrounds, editing, and publishing. Or maybe even music: lyrics, instruments, a recording studio, equipment, distributors. What does the makeup of all these hobbies have in common? There are a lot of "small pieces" that need to come together in order to bring the hobby to life, or rather "create" the hobbies. *Complexity Theory* is composed from the same basic idea: a few "small things" create a "big thing". In short, Complexity Theory states that when given a few simple rules, the emergence of complex systems arise. These rules typically involve some sort of individual and their interactions with other individuals as well as the surrounding environment.

Where Do We See It?

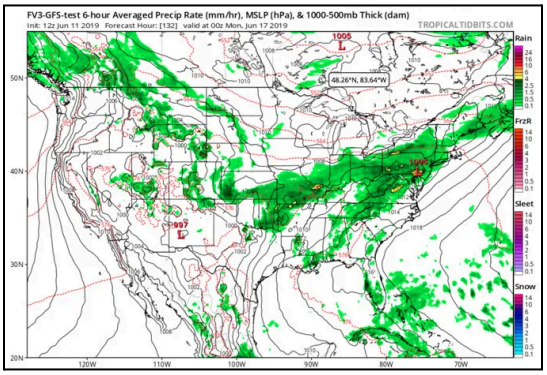


Complexity emerges in all areas of life. Starting most simply with you! Every living being is created from singular cells that on their own, can not do much. However, when placed together in a system, we see the emergence of beings that are capable of thinking, moving, foraging and creating. Even cells themselves, are composed of organelles which on their own don't do much, but when in a system together, cells that are capable of the very basics of life emerge. We see it in our economy too! A few basic rules defined by people (i.e., supply and demand) lead to the numerous intricacies of our economy. Have you ever seen a flock of birds before? Did you ever wonder how the group as a whole was able to move in such an organized pattern? The answer; complexity theory! Each bird follows some simple rules: don't fly too close to the bird in front of me, move a little to the side to get away from the bird next to me, go in the same direction as those around me, etc. For a single individual, this may result in seemingly random, or even uniform, movements. However, when multiple individuals are placed

together in the environment, the result is a flocking pattern that arises without the need for communication between any of the individuals.

Understanding Complexity Theory?

Why do we care about complexity? It's cool, but what benefit does it serve to study? Especially if the rule sets systems are defined by often leading to unpredicted answers than what we may have originally assumed. Prediction is exactly why we care about complexity theory. Many problems we encounter today have some layer of complexity to them— from finding the best route on your GPS to get to work, to understanding weather patterns to predict future events, and even artificial intelligence! These simple rule sets define many of the most intricate things we encounter daily, so if we are able to make predictions about them, it could be the difference between showing up to work five minutes late drenched in rain and showing up five minutes early with an umbrella. Predictions are usually made based on models of complex systems. The best part, you can make these models yourself!



How can we experiment with it?

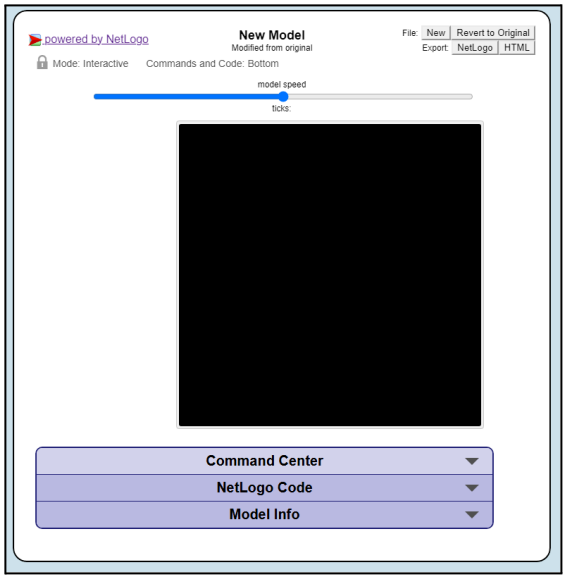
You may be wondering to yourself, "How can we possibly model complex systems?" Computers! And a little bit of coding. When it comes to modeling complex systems specifically, we want to use something known as "Agent-Based Modeling". In agent-based modeling, there is an environment which consists of different agents. The agents (think "players" in a game) have a basic set of rules, and are capable of interacting with each other and their environment. Based on the set of rules you define for them, and the environment you create, you can run a simulation and see what end results you get. A popular agent-based modeling language is called NetLogo. NetLogo has two versions, a desktop and web-based version. For purposes of accessibility, this guide will make use of the web-based version. If you've never coded before, don't worry!

NetLogo is very "people" friendly in the way it is written, and is a great starting point for starting off your coding journey!

How does NetLogo work?

So, how do you get started? First, go to your web browser and search "netlogo", and then click on the link for "NetLogo Web" (www.netlogoweb.org). Then, for us to get started, click the "New" button in the upper right hand corner of the screen.

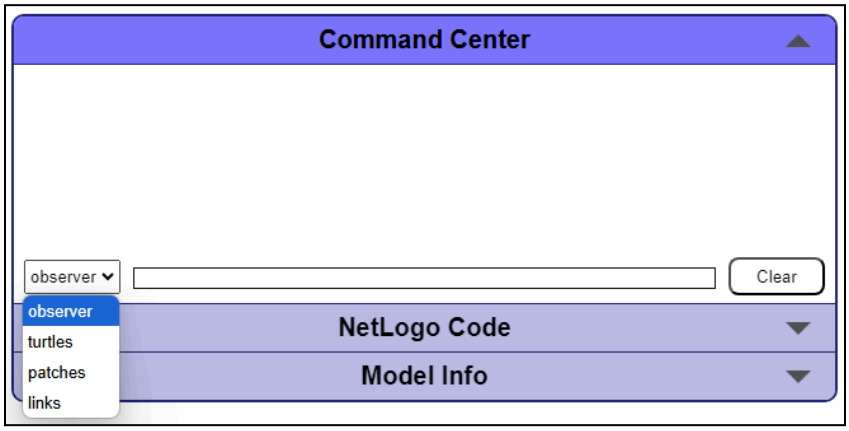
From here, there are a few places you should familiarize yourself with. First is the environment that you are working in. There are several key parts to go over. The window in the middle of your screen is the environment that our systems will be simulated in. This is also



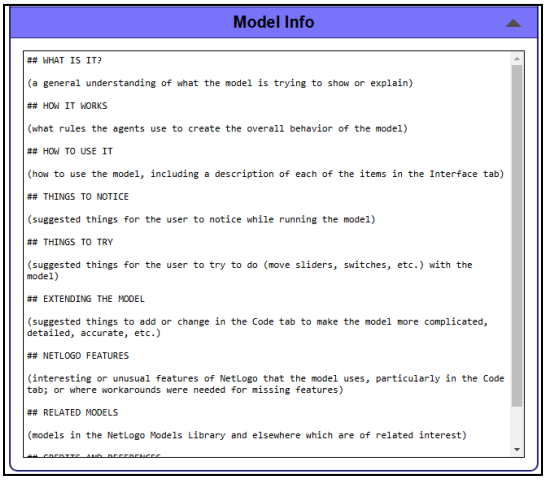
where all of the agents that we create will appear in. Above that window is a slider titled "model speed". This slider allows us to control the tick rate of our simulation, or in other words, the speed of the simulation.

Underneath the window are three drop-down tabs, "Command Center", "NetLogo Code", and "Model Info". The Command Center is an area where you can type brief sections of code to cause an immediate reaction within the environment. There is no need to save your code or compile it before running (and if you're not sure what 'compile' means, don't worry, that will be covered next!). Within the command center, you may notice a button that says "observer" on it. If

you click on it, you will be given different options such as "turtles", "patches", and "links". To get started, keep the option selected to the observer. If you select "turtles", "patches", or "links", then you are communicating directly with one of the agents. When the button is set to the observer, you are talking to the environment as a whole. The code we will eventually write is from the observer perspective, so keeping the command center set to the observer is a good way to test out ideas you may want to include in your code.



NetLogo Code is the area where you will type the rules that will define your simulation using the NetLogo programming language. In this section, any code you type will be locally saved to your browser during your session, however, it is a good idea to save your code to a backup file every iteration to ensure there is always a back-up! You can do this either by simply copying and pasting your code in a separate document (like a Google Doc) or by using the export buttons in the top right corner of your screen, just underneath where the "New" button is from earlier. If you plan to use the export button, you can either export a NetLogo file or HTML file. In order to restore an older version of your model, make sure your export file is in the **.nlogo** format (downloaded by clicking the "NetLogo" export button) this way you can upload the file to NetLogo Web at a later time to continue editing.



Finally is the Model Info. This is a nice area for documentation of your simulation, or in other words, where you can provide a brief description of what your simulation does and how it works, as well as any references, to anyone who may use it in the future. For larger models especially, this is an

important section of your simulation to have filled out, as the larger the model, the more difficult it can be to follow.

To the top left of the screen, you will see a gray lock. Clicking this lock will place you into an editing mode for the section outside of the window. In this new area, which will be highlighted in green, you can add various types of user interfaces such as buttons and sliders, to make it easier for anyone to interact with any parameters of your simulation. Additionally, this will allow you to create buttons that will allow a user to easily setup, start, and stop the model. This editing mode also allows you to change the Model Info section.

As they always say, practice makes perfect. To help ease you into the NetLogo programming language, there are a series of mini-projects that will culminate in a model that can create complex art in less than 20 lines of code and using only two rules! In addition, these projects will make use of all the basic components of NetLogo mentioned above, giving you a baseline from which you can continue to experiment with and create your own models!

Project 1: Draw a Line

Getting started, the goal is simple: draw a line. And to do so, we don't even need to create a file! Everything can be easily done from the Command Center! To begin, if you had a piece of paper and a pen, ask yourself, "How do I draw a line?" Be specific, making sure to clearly label the steps in your head. Got it? Perfect! Everyone may have a slightly different way of describing it, but as long as your steps follow those below, you're on the right track!

Step 1) Pick up the pen
Step 2) Place the pen **down** on the sheet of paper
Step 3) Move the pen **forward** on the sheet of paper
Step 4) Pick the pen **up** from the sheet of paper
Step 5) Put the pen back

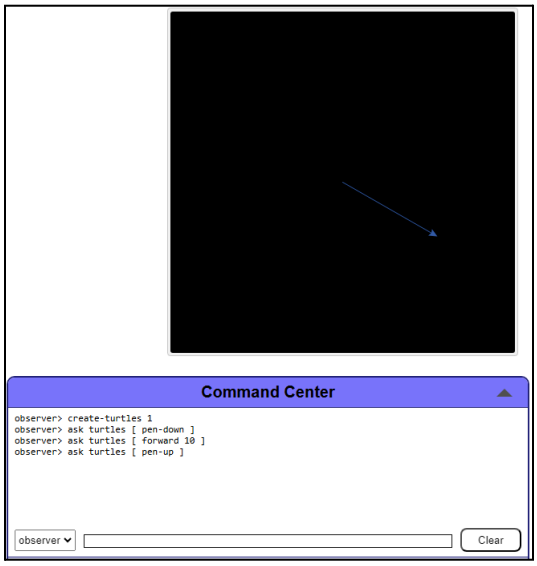
This may seem silly to do at first, however, whenever you approach a task that you want to recreate in code, it's important to be able to break it down into simple steps. As silly as it seems, drawing a line isn't an easy task for a computer! It has no idea how to do it. We need to be very clear in telling it how it can draw a line, otherwise, it won't! Let's break down our steps into code the computer can understand.

In NetLogo, an agent called the "turtle" is capable of drawing on the screen! They are also capable of much more, and are not the only agent type. For the projects in this guide, we will only need the turtle agent type. In order to interact with the turtles, we first need to have some turtles on the screen. In NetLogo, we can simply do this by typing create-turtles #. The hashtag represents the amount of turtles you want to create. For now, let's create just one turtle. Type this into the command center (make sure you are on the "observer" mode) and then look at the window above. You should now see a turtle!

Great…. now what? Think back to the series of steps from before. After picking up the pen, you need to place it **down** on the paper, move it **forward**, and then pick it back **up**. Fortunately, NetLogo has a series of commands that will allow the turtle to do this! In the command center, type one at a time:

```
ask turtles [ pen-down ]
ask turtles [ forward 10 ]
ask turtles [ pen-up ]
```

Let's break down these lines of code. Whenever we want the turtles to do something, we should be polite to them and *ask* them to do so. Thus, we always start with *ask turtles*. The [ ] are used to enclose what we are asking the turtles to do. *pen-down* tells the turtles to put their pen down, meaning wherever they move, a trail will be left behind them. *forward 10* tells the turtle to move forwards 10 units in the environment. *pen-up* is the opposite of *pen-down*, having the turtles pick up their pen and preventing a trail from being left whenever they move somewhere. If you look back at the window, you should see the line drawn!



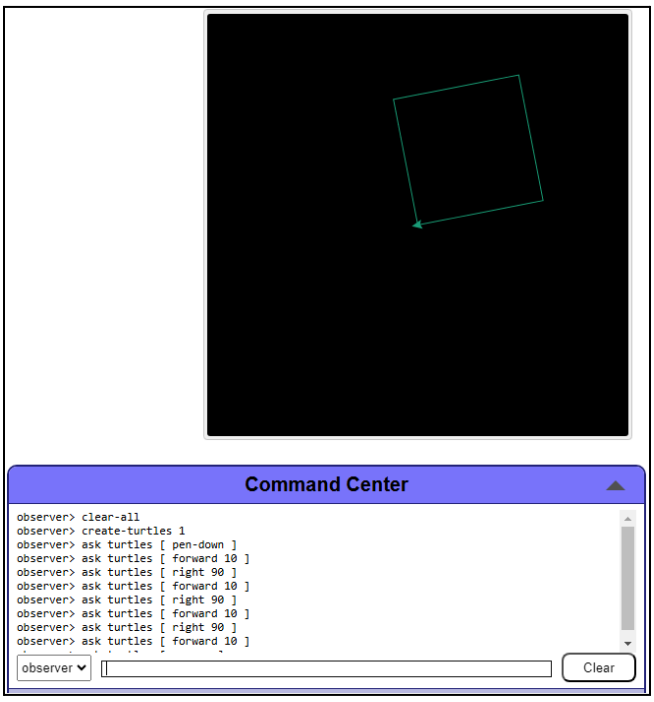
Congratulations!! You've completed Project 1!!

Project 2: Draw a Square

Let's start Project 2 the same way we started Project 1. What are the steps in drawing a square? Remember to be specific and clear. It's always better to have extra simple steps than too few complicated steps. Similar to before, as long as your steps line up with the ones below, you're on the right track!

Step 1) Pick up the pen
Step 2) Place the pen **down** on the sheet of paper
Step 3) Move the pen **forward** on the sheet of paper
Step 4) Turn **right 90** degrees
Step 5) Move **forward** the same amount
Step 6) Turn **right 90** degrees
Step 7) Move **forward** the same amount
Step 8) Turn **right 90** degrees
Step 9) Move **forward** the same amount
Step 10) Pick the pen **up** from the sheet of paper
Step 11) Put the pen back

A square is really just drawing four separate lines at 90 degree angles to each other. As of right now, you have all the tools you need to do this from the command center except for how to turn your turtle. To do so, type ask turtles [right 90]. Additionally, before trying to draw the square, you can also type clear-all, to clear the line and all turtles from Project 1. Give it a shot, and then compare your results to the picture to the right!

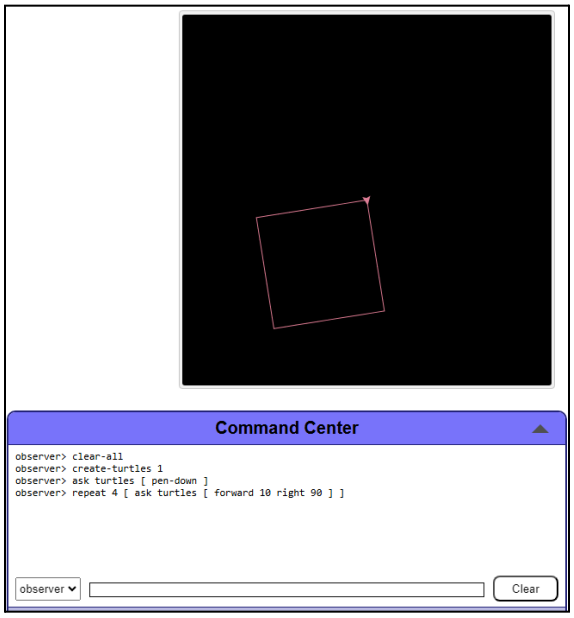


Awesome job! Now, that was a lot of commands we had to type. Why? Because we had a lot of steps! We should try to simplify our steps to see if we can reduce how many there are. Take a moment, and look at the steps from before. Is there any way to simplify them? (Hint: look for patterns, commonalities, any repeats, etc.) Once you have an idea, write it out and compare it with the steps below!

Step 1) Pick up the pen
Step 2) Place the pen **down** on the sheet of paper
Step 3) Move the pen **forward** on the sheet of paper
Step 4) Turn **right 90** degrees
Step 5) Repeat steps 3 & 4 three more times
Step 6) Pick the pen **up** from the sheet of paper
Step 7) Put the pen back

All we had to do was combine steps 5 through 9 into a singular one that tells us to **repeat** a certain action. In NetLogo, we can do this using the **repeat** command. After clearing the screen using *clear-all*, type the following into the command center:

```
create-turtles 1
ask turtles [ pen-down ]
repeat 4 [ ask turtles [ forward 10 right 90 ] ]
```



Like before, let's break down this code. *create-turtles 1* adds a turtle to the window like before. *ask turtles [pen-down]* has the turtles put their pen down on the paper so they leave a trail wherever they go. *repeat 4* indicates that the code inside of the [ ] will be repeated four times in a row. Inside, *ask turtles [forward 10 right 90]* has the turtles move forward 10 units and then turn to the right 90 degrees. This is a more concise way to have the turtles move and turn as well, since we ask them multiple things at the same time! Think, when you order food at a restaurant, you don't

say: "Can I order a burger? Can I order fries? Can I order a drink?" You would instead say: "Can I order a burger, fries, and a drink?" Combining multiple asks in the same question to the turtles will help the code we write later be more readable. Once you've tried typing the code, compare it with the picture above. After doing so, congratulations on completing the second project!!

Project 3: Draw Three Squares

Okay, for Project 3, we're going to take our work to the next level! We're going to write our first block of code inside of the NetLogo Code section. As we did before, we're going to start by listing out the series of steps we need to take to draw three squares. Take a moment to do so, and then compare it with the list of steps down below. Since we've already described how to break down a square in detail, focus more on the new task of drawing *three* squares. Be specific to that task.

Step 1) Pick up the pen
Step 2) Place the pen **down** on the sheet of paper
Step 3) Draw a square
Step 4) Pick the pen **up** from the sheet of paper
Step 5) Move the pen to a new place on the sheet of paper
Step 6) Place the pen **down** on the sheet of paper
Step 7) Draw a square
Step 8) Pick the pen **up** from the sheet of paper
Step 9) Move the pen to a new place on the sheet of paper
Step 10) Place the pen **down** on the sheet of paper
Step 11) Draw a square
Step 12) Pick the pen **up** from the sheet of paper
Step 13) Put the pen back

Like before, this is a lot of steps. Is there any way to simplify this. Give it a shot, and see if you can make it simpler. Then, compare it with the steps on the next page.

Step 1) Pick up the pen

Step 2) Place the pen **down** on the sheet of paper

Step 3) Draw a square

Step 4) Pick the pen **up** from the sheet of paper

Step 5) Repeat steps 2, 3 & 4 two more times

Step 6) Put the pen back

Awesome! We have a game plan now for how to proceed with our code, so let's get started. Below will be a block of code that will complete the task above. The sections of the code will be described below. Read it slowly and take time.

```
;; Project 3

to setup ;; how to setup the window
  clear-all
  create-turtles 1
end

to go
  repeat 3 [ ;; repeat allows us to draw three squares
    ask turtles [ ;; giving instructions to the turtle

      pen-down ;; place the pen down back on the paper

      drawSquare ;; Draw one square

      pen-up ;; pick the pen up from the paper

      setxy random-xcor random-ycor ;; move pen to different position on the page

    ]
  ]
end

to drawSquare
  repeat 4 [ ask turtles [ forward 10 right 90 ] ] ;; instructions to draw one square
end
```

First, what are the ;;. The ;; represents the start of a comment, which is just writing you put in your code to help you or anyone who looks at it understand what is going on. Everything after it is ignored.

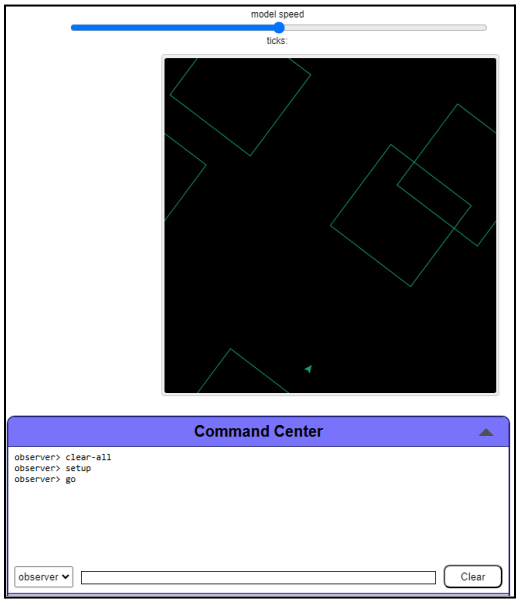
In the code, we have what are called *functions*. There are three functions that are defined: setup, go, drawSquare. What is a function? A function is a block of code that does a specific task. You can think of them like when we use *forward 10* for the turtle, just instead of them already being built into NetLogo, we have to create them ourselves! Functions can be used inside of other functions! To create your own function, you start with "to" followed by the name of the

function you're going to create. Then, you write out the block of code that will run whenever you use the function, and once you write all of it, you close off the function using "end". In NetLogo, whenever creating a model it is traditional to create a setup function and a go function. setup usually wipes the slate clean and creates the starting conditions needed for the simulation to run. go defines the main simulation. This is what you want to have happen.

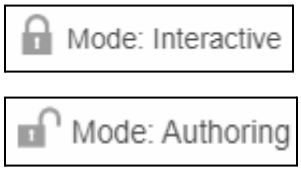
For Project 3, we start with the setup function. It clears the screen of any previous models, and then creates turtles in the window. When we are ready to run the model, we use the go function. It tells us that we're going to repeat a block of code three times in a row. This block tells us we're going to first ask the turtles to place their pen down, then draw a square, pick up their pen, and finally move to a random xy-coordinate in the window. After this block of code has been run three times, the go function ends. Finally, we also defined a drawSquare function which was used inside of go. The drawSquare function uses the results from Project 2 to describe how to draw one square.

Try typing the code inside of the NetLogo Code section and get used to the feeling of typing commands outside of the Command Center. In order to check if you typed your code properly (and before you'll be able to run it!) you need to *compile* it. All this means is that you are asking the computer to check over your work and make sure you have no reading errors in the program's language. If there are any, the computer will tell you about them. The "Recompile Code" button is right underneath the NetLogo Code section when expanded.

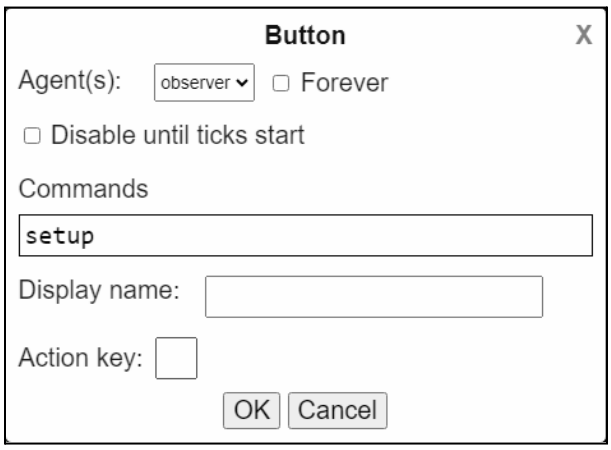
Once you compile the code, it's time to run the program! Only one question is left… how? In NetLogo there are two ways to do so. First, you can run the code through the Command Center. Inside of the command center, you can type the name of the functions you've custom made, i.e. *setup* and *go*. Try first typing *setup* in the command center, followed by *go*. Your window should now have three squares drawn in it!

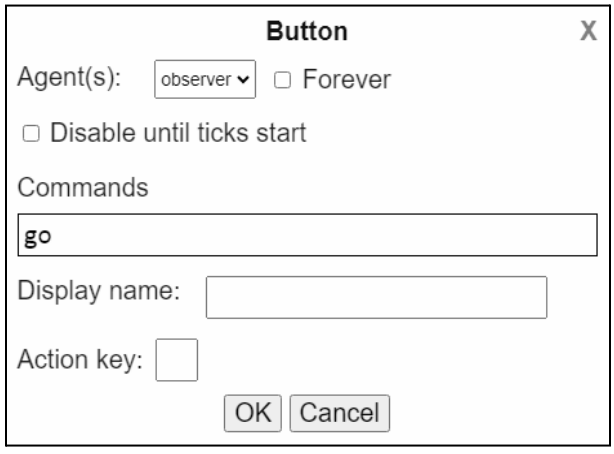


You may notice that some of the squares don't look fully drawn, and in other places it looks like there is barely any square at all. This is okay! In NetLogo, the window automatically wraps around itself by default. This will be useful for us later in Project 5.



The final way to run your code through NetLogo would be by using buttons! To add a button, we have to first click the gray lock icon in the top left corner of the screen. This will take us from "Interactive Mode" to "Authoring Mode". In authoring mode, part of the screen will now become green. In this part of the screen, you will be able to add buttons for the setup and go of the simulation. To add a button, start by right clicking anywhere within the green section. A drop down list will appear, and contain all of the different types of user interfaces you can add to the section. We will select the first option, "Create Button". Upon doing so, a new popup will appear allowing you to enter information for the button. Make sure the agent(s) are set to "Observer", and then in the box labeled "Commands", type our first command, *setup*. Then click "OK", and you will have created your first button! You can then drag the button around the screen and place it anywhere you would like within the green section. Design it however you like! We're going to repeat this process to add a button for go, with the only difference being in the "Commands" box, we will enter *go*.

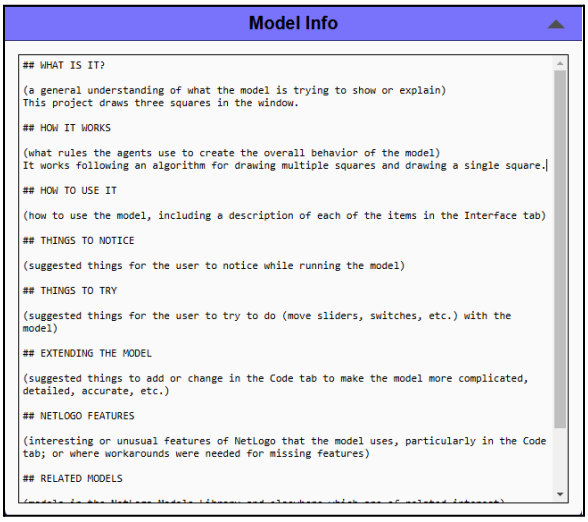


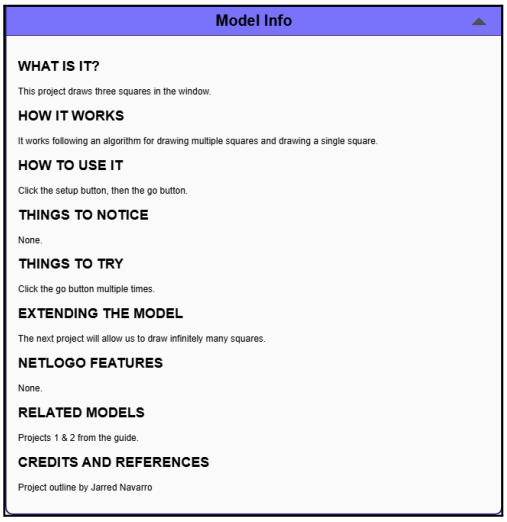


After everything is done, you can click the gray lock button again to return to "Interactive" mode. To run your model, first click the setup button, followed by the go button. And that's it! You should see three squares get drawn to your screen. If you do, you finished Project 3!

A final nice touch to add would be within the "Module Info" section. Since we have officially created our first NetLogo model, it is important to include some documentation here so any future users will know what our model does. To edit the text inside this section, click the gray lock to enter "Authoring" mode, and then open the dropdown tab for "Module Info". Follow the pre-provided outline, and fill out as much information as you can regarding the model. Once you're done, click the gray lock again to return to "Interactive" mode.

Last but not least, make sure you save all of your hard work! To export your work, go to the top right of the screen, and next to where it says "Export", click the button for "NetLogo". This will download a .nlogo file that you can upload to NetLogo Web in the future so you can continue editing this project, or easily share it with friends! You could also export your project as an HTML file, which would allow you to open it via any web browser and run your module, however, you **will not** be able to import your project back into NetLogo Web to continue editing it. You could edit the model from the HTML site that gets created, but any edits you make will not be saved locally to the browser. Choose whichever file type feels the best for you! Or both!





Project 4: Create rainbow square art

So far, so good! You're becoming a master of NetLogo. Now it's time to take it one step further again! This time our goal will be to create rainbow square art, or in other words, we want the model to draw an infinite amount of squares, but each square's color is randomly chosen. Like the previous three projects, take a moment to think of how you would do this. Remember to

be specific, and clearly outline all of your steps. Once you've done so, check with the steps on the next page!

Step 1) Pick up the pen

Step 2) Choose a color

Step 3) Place the pen **down** on the sheet of paper

Step 4) Draw a square

Step 5) Pick the pen **up** from the sheet of paper

Step 6) Choose a new spot on the page

Step 7) Repeat steps 2, 3, 4, 5 & 6 **forever**

Lets walk through each step and see if we know how to do it with our knowledge so far in NetLogo. Try going through it on your own at first, and then check below to see if you got everything that we know!

Step 1) Pick up the pen – This is just a person thing!

Step 2) Choose a color – We don't know just yet, but we will soon!

Step 3) Place the pen **down** on the sheet of paper – ask turtles [pen-down]

Step 4) Draw a square – We have a function for this!

Step 5) Pick the pen **up** from the sheet of paper – ask turtles [pen-up]

Step 6) Choose a new spot on the page – We have the code for this from Project 3

Step 7) Repeat steps 2, 3, 4, 5 & 6 **forever** – We don't have this yet!

Most of what we learned so far applies again, which is perfect! There's only a few new things we need to learn: how to choose a color and how to do something forever. Let's run through those now.

In NetLogo, as with all other things we want the turtles to do, in order to change their color we have to ask them. Return to the command center, clear the screen, and add a turtle to the environment. Once you do so, type the following: ask turtles [set color blue]. After doing this, type the same line again but try a different color, like red, green, orange, etc. If you want, try adding or subtracting values from the color to change its shade. Adding will make it a lighter shade of the color and subtracting a number will make it a darker shade. (Ex: blue - 2)

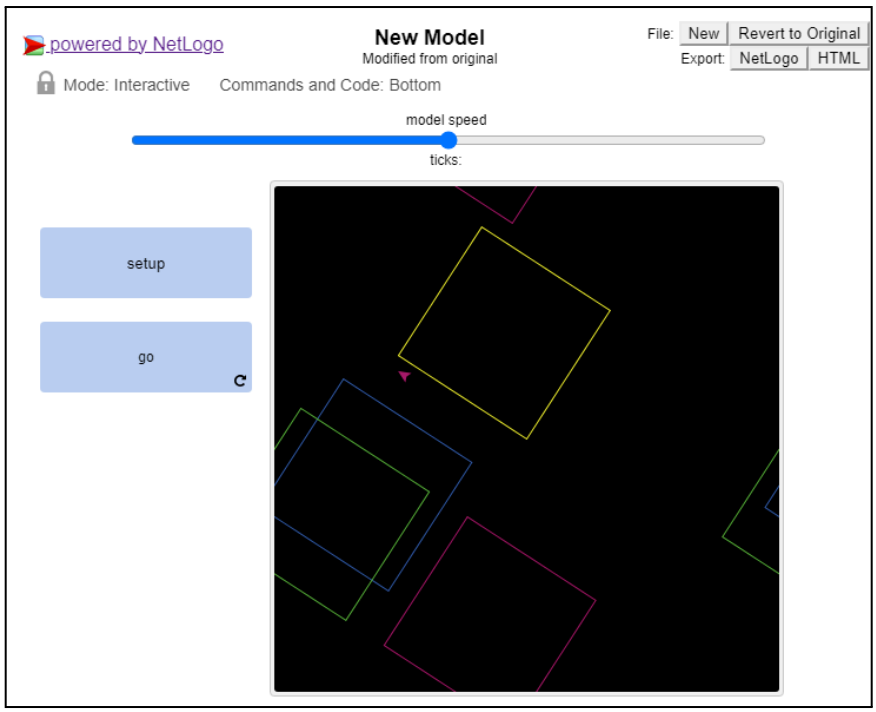
If you are looking to just set the color one time and never change it, you're all set! However, sometimes it would be useful to change the color randomly. To do this, we will need to use a slightly different line of code: ask turtles [set color one-of [blue red green]]. What does this mean? The main difference is the *one-of [blue red green]*. What this section of the code does is tell the computer it will randomly choose one of the colors from the three listed in the [ ].

Using this information, let's update the code from Project 3 to match our steps from before. Give it a try, and then compare your code with that of the picture below. If they are similar, then you are on the right track!

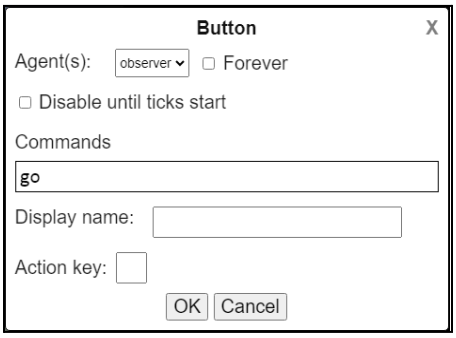
```
;; Project 4

to setup ;; how to setup the window
  clear-all
  create-turtles 1
end

to go

  ask turtles [ ;; giving instructions to the turtle

    pen-down ;; place the pen down back on the paper

    set color one-of [ red orange yellow blue green magenta ] ;; assign a random color

    drawSquare ;; Draw one square

    pen-up ;; pick the pen up from the paper

    setxy random-xcor random-ycor ;; move pen to different position on the page

  ]

end

to drawSquare
  repeat 4 [ ask turtles [ forward 10 right 90 ] ] ;; instructions to draw one square
end
```

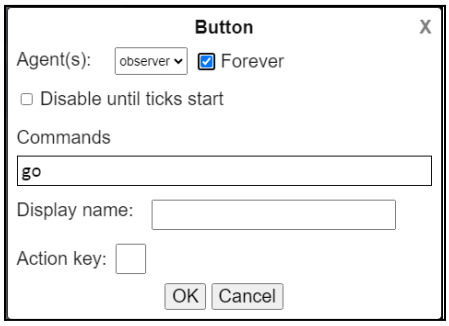
What changed from Project 3? In Project 3, we used a *repeat* in order to draw three squares to the screen. However, since our task this time only needs us to draw one colored square at a time, we were able to remove that part of the code, as well as the [ ] that held what we wanted repeated. The second difference between Project 3's code and Project 4's is that we used the *set color one-of* alongside a list of colors. NetLogo has a total of 14 primitive colors you can choose from including: gray, red, orange, brown, yellow, green, lime, turquoise, cyan, sky, blue, violet, magenta, pink. The list you want your model to randomly choose a color from can include as few or as many of these colors as you would like, and can include as many different shades of the colors as you would like! The limit is your imagination! Aside from those two changes, the

code is exactly the same as Project 3's. Even the buttons we created in Project 3 can be reused! However, in order for any of the changes in the code you made to go through, you have to make sure you compile the code again. Once you've done so, click the go button a few times and you should begin to see squares get drawn in different colors at different places in the window!

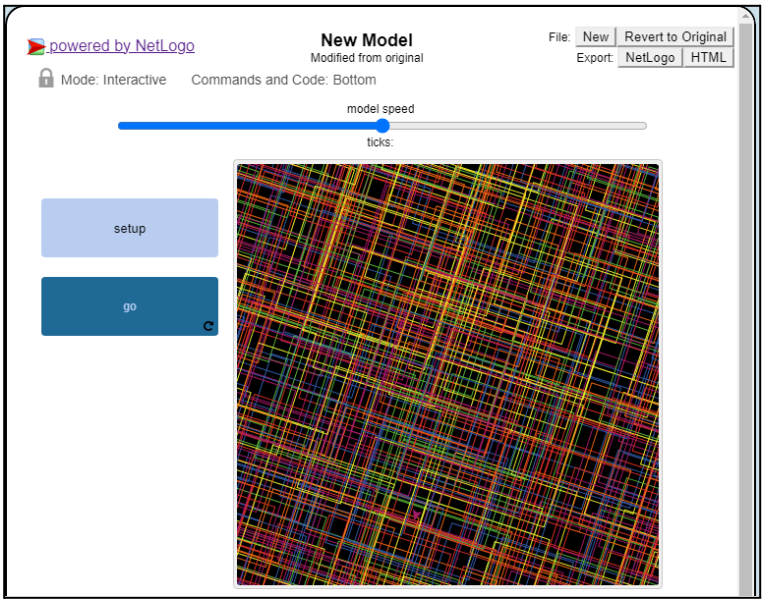


The only question left is, "How do we make the program run forever?" As of right now, if we were to click our go button, it will only draw one square to the screen. Last I checked, one is much less than infinity. So, how are we getting there? The answer is fairly simple! First, we will need to click the gray lock in the top right of the screen to enter "Authoring" mode. Once in "Authoring" mode, you need to right click on the go button, and then click "Edit". In the window for the go button, you should see a checkbox, with the word "Forever" next to it. Make sure you check that box on, and then click "OK" at the bottom. What did this do?





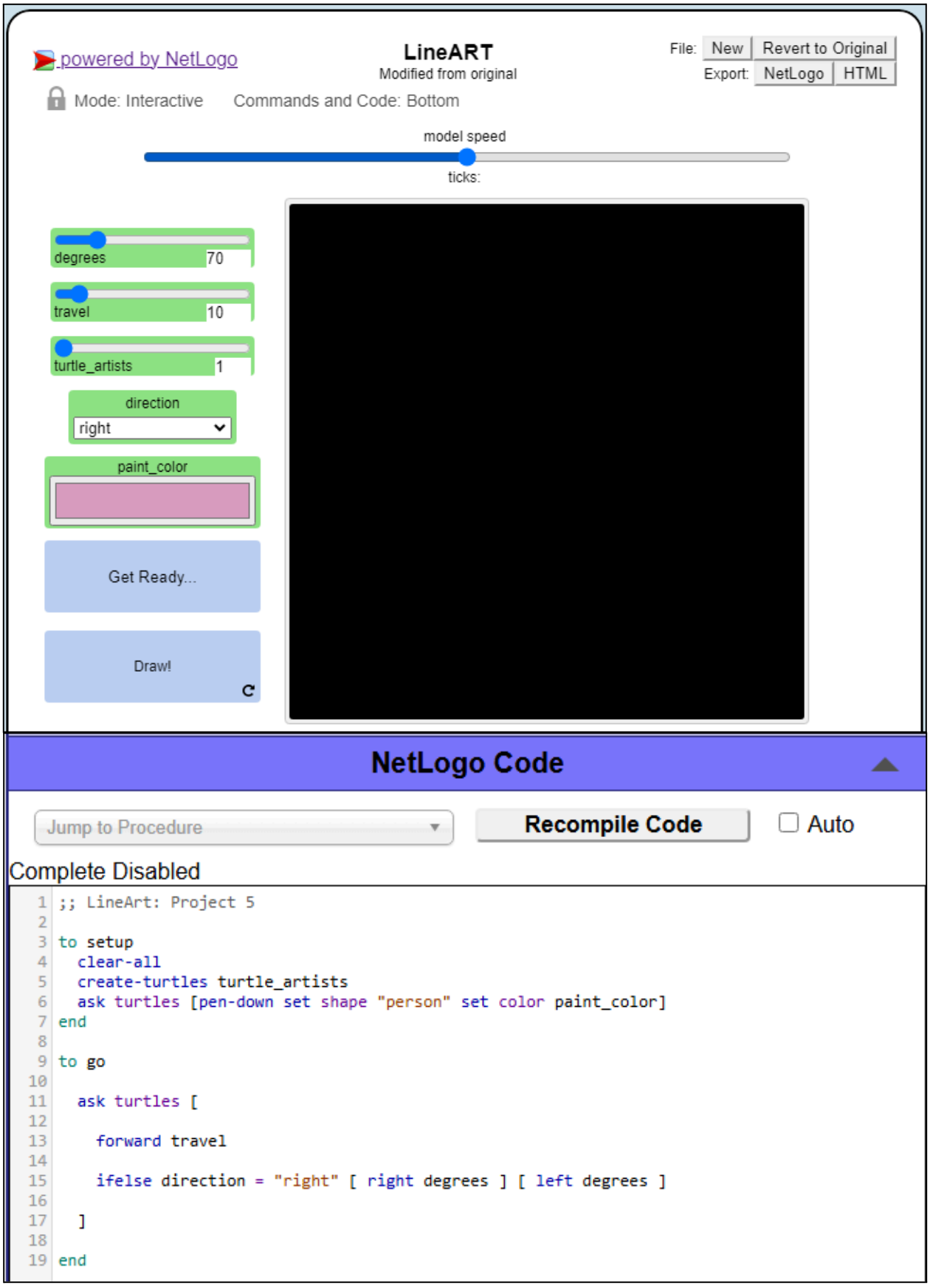
By enabling this box, you allowed the go function to run forever so long as the button is pressed. In other words, once you click the go button, the go function repeats until you click the button again. Give it a shot, and then see the art that begins to form on your screen! Once you have squares being drawn, try messing around with the code a little more and make it more unique to you. Maybe you want the squares to be a different size. Maybe you want to make a shape other than squares. Whatever the twist, here is your challenge: Try to make it in the code! And with that, congratulations on completing Project 4!



Project 5: LINE*ARt*

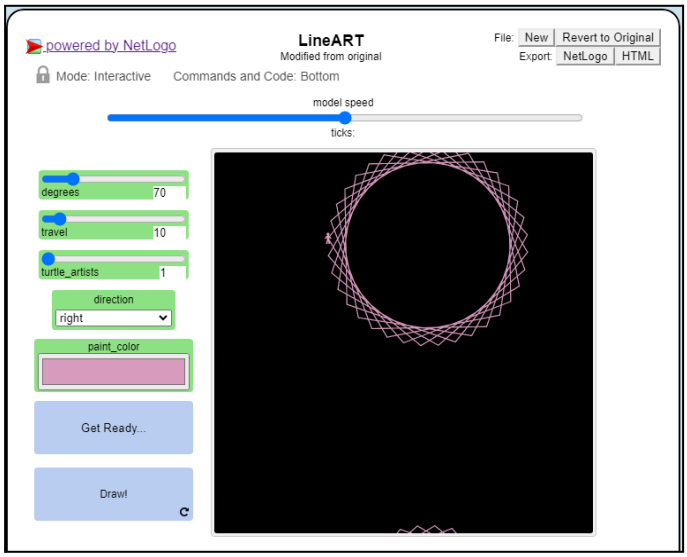
You have done an amazing job so far completing these first four projects! For the final project, it takes everything that you have learned so far and pushes it a step further again. However this time, instead of walking you through step by step how to code it, I want you to try it on your own. A picture of the code and all necessary parts of the model will be included below. Now, you are at a point where you have mastered the basics of NetLogo, and now must master the ability to read, decipher, and understand the code. Additionally, you also should begin practicing utilizing documentation on NetLogo to help expand your skill set. Unsure what the command ifelse does? Look it up! Want to change the shape of the turtle? Reference the documentation! Knowing how to look up the documentation and interpret it as well as understand someone else's code are important parts of being able to continuously grow and develop your skills within not just NetLogo, but any programming language.

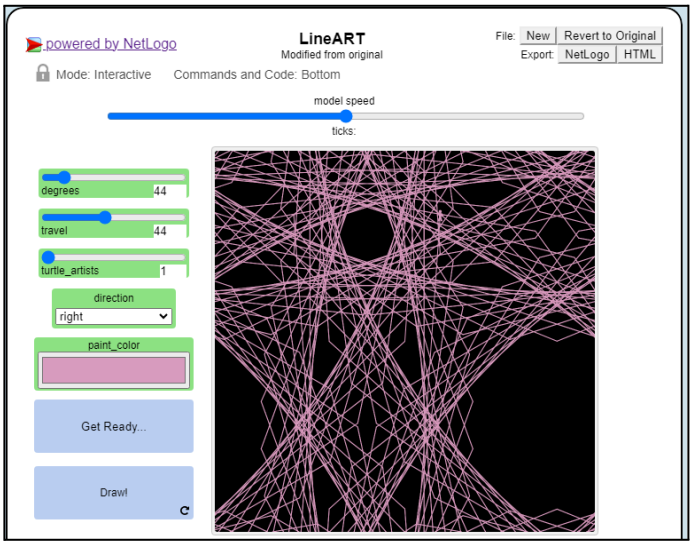
After attempting to analyze the code, write out your understanding of it and compare it to my breakdown of the code. As always, if you're close, then you're moving in the right direction!

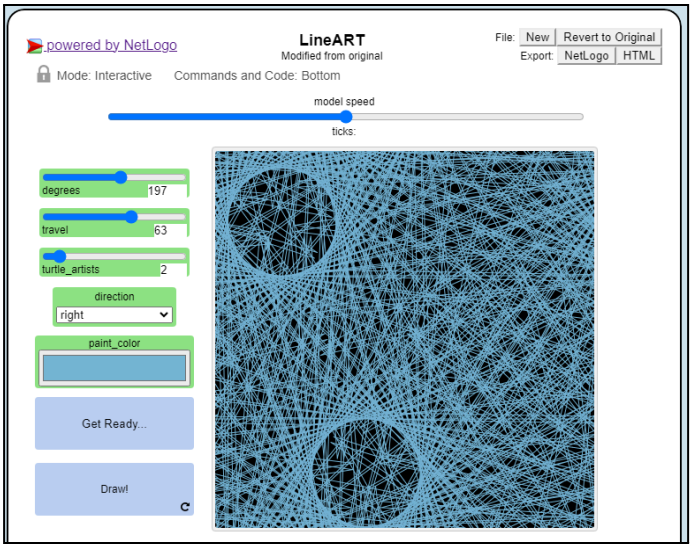


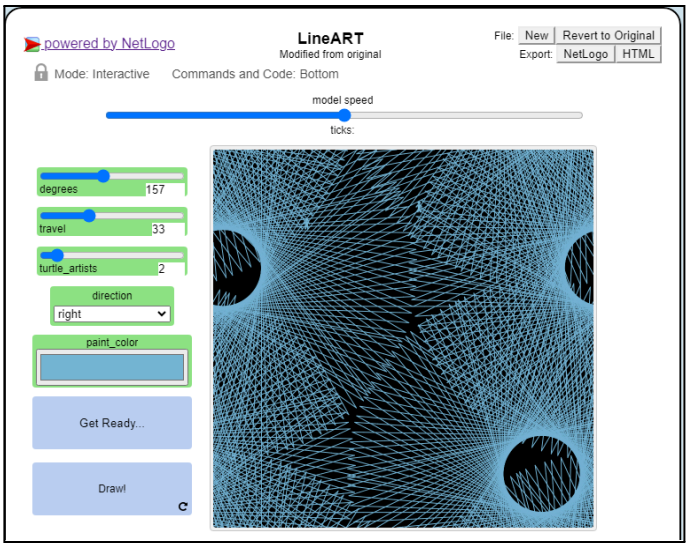
Notes: turtle_artists represents how many turtles you want to create, travel represents how many units you want the turtles to move, and degrees represents how many degrees you want the turtle to turn. The buttons and code work hand in hand for this project.

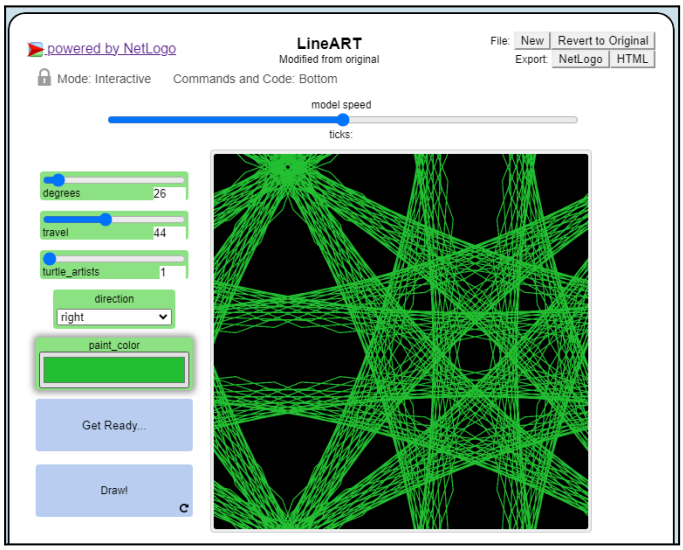
Some outputs of the program can be seen below:

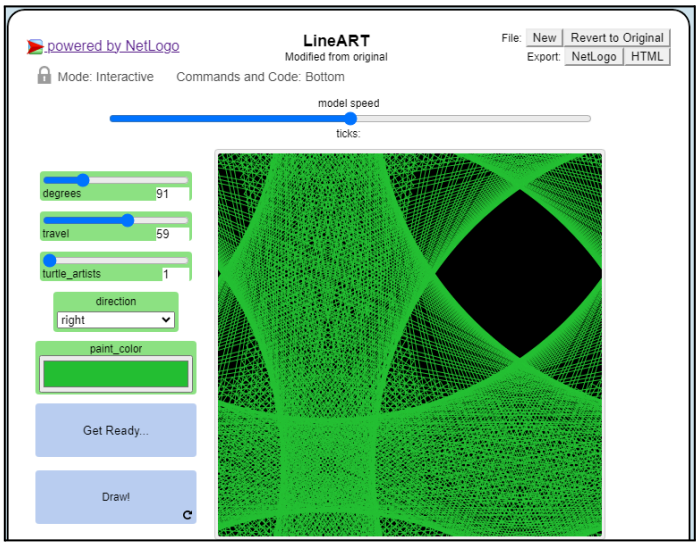












Below are guiding questions you can use when trying to understand someone else's work. Are these the only questions you can ask? Absolutely not! But they can provide a solid foundation from which to build on.

Q: What is the goal of this model?
A: To create art using lines and angles.

Q: Are there any variables I can change?
A: Yes! degrees, travel, turtle_artists, direction, and paint_color are all variables the user can change using the buttons in the model.

Q: How does the model run?
A: (See below…)

The model starts with the setup, which involves clearing the window, creating an amount of turtles, specified the value of turtle_artists (i.e., the user chooses how many turtles are made), and then asks the turtles to place their pens down, change their shape to a person, and set their color to match that of the user's choice as well.

Then, the model uses the go function, which asks the turtles to move forward travel units, i.e. the amount of distance specified by the user using the buttons. Then, according to the documentation, the computer checks if the user selected the direction to be right or left. If they choose "right", then the code in the first [ ] runs, turning the turtles to the right. Otherwise, the code in the next set of [ ] runs and the turtles turn left.

Inspecting the model a little further, and based on my own experimenting, the setup button is represented by the one named "Get Ready…", and the go button is represented by "Draw!" Additionally, it is clear that the go button is set to run for forever based on the little arrow in the bottom right corner of the button.

The most important part of your journey moving forward is to be courageous and curious. Look up things you aren't sure about, use documentation, try to make your craziest ideas. There is much more to NetLogo than discussed in this guide, but you now have the basic tools to get started on your next project. So… congratulations on completing Project 5, and good luck! :)