Building a Simple Rock-Paper-Scissors Model in NetLogo

In this activity, we will build a simple NetLogo model in which the turtle agents play a simplified form of Rock-Paper-Scissors. As previously discussed, even this simple game has parallels in natural world, including ecosystems of three strains of *E. coli* bacteria, as well as the three different types of side-blotched lizard males. In the case of *E. coli*, the parallel is very close, and this type of model matches the real-world observations very well.

In later exercises, we will add additional features to this model. But for now, the characteristics are as follows:

- The terrain will be completely populated by turtle agents i.e. there is one turtle agent per patch.
- Turtles will not move.
- Each turtle can only play against those turtles directly adjacent to it i.e. those located immediately to the North, East, West, or South of it.
- Each turtle will start out with a strategy of rock, paper, or scissors, selected randomly. It will maintain this strategy until it is defeated, at which point it adopts the strategy of the victor. Whenever it changes to a new strategy, it will continue with the new strategy until it is once again defeated, and once again adopts the victor's strategy.

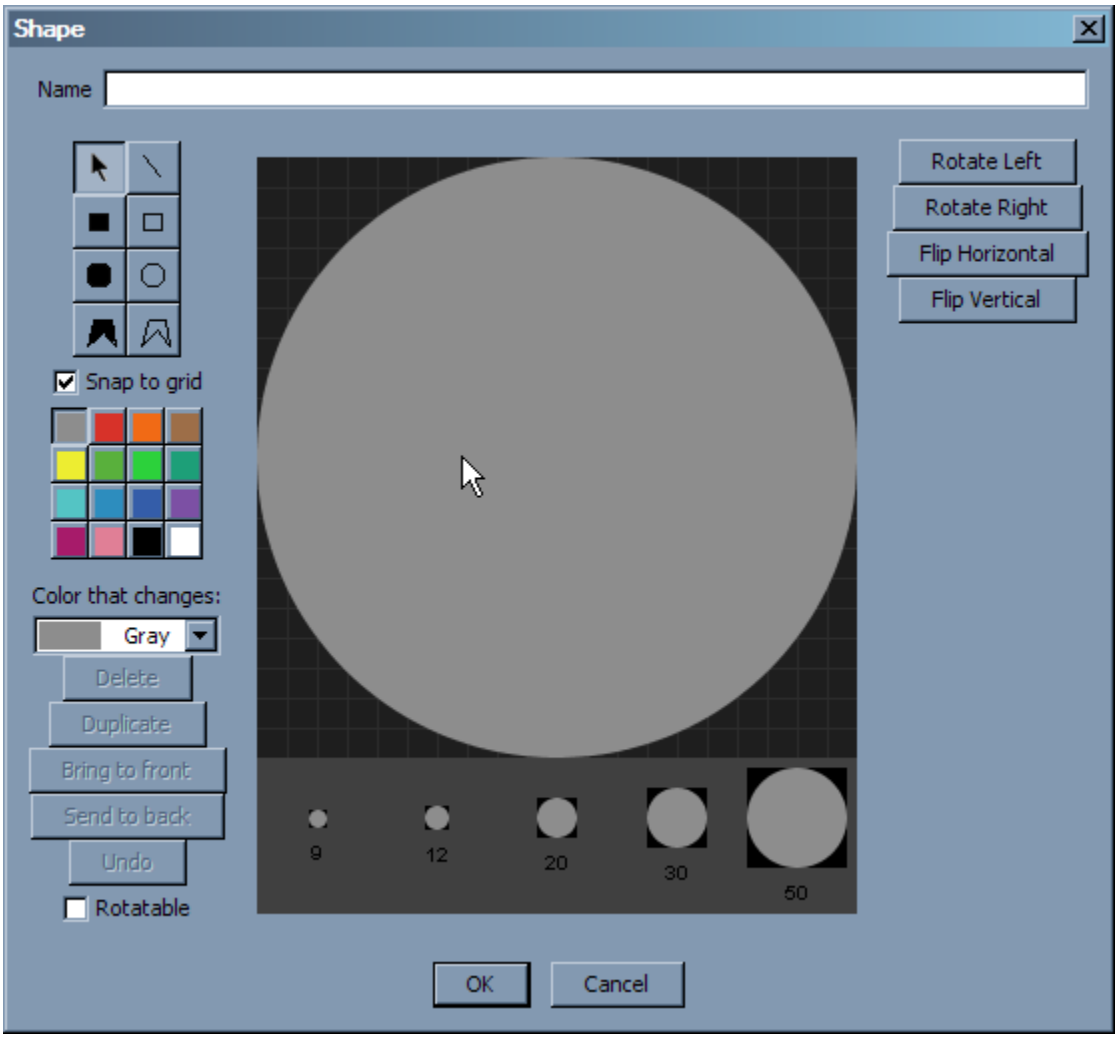
Setting Up the Model

Rather than make this model specifically about the three *E. coli* strains, or the three types of side-blotched lizard males, let's simply have three generic species: rocks, papers, and scissors. In NetLogo, we do this with three breed statements (at the top of the **Procedures** tab):

```
breed [ rocks ]
breed [ papers ]
breed [ scissors ]
```

(In this document, any new code added at each step will be italicized; code that was created in a previous step will not.)

To make things very simple, let's make the turtle shapes circles, colored red, green, and blue (respectively). Select the **Tools/Turtle Shapes Editor** menu option, and find the circle shape. We will use this shape but to make things easier in our code, we will create three special versions of this shape: one colored red, one colored green, and one colored blue. We do this by selecting the circle shape, and then pressing the **Duplicate** button. We then see this:



Give this shape a new name (I suggest rocks-circle or rock), then click on the big gray circle. When it is selected, click on the red tile in the color palette to the left. This will change the color of the circle to red. Click **OK**, and our new shape appears in the shapes list.

Repeat the above process duplicating the original circle, giving a new name to the new shape, and changing the color to create blue and green circle shapes for papers and scissors (respectively). When this is done, close the **Turtle Shapes Editor** window.

Now our model needs a setup procedure:

```
to setup

end
```

(Note that the above shows a blank line in the procedure; this is where we'll start writing the statements that will instruct NetLogo how to set up the model.)

We will begin the setup by clearing everything, and then telling NetLogo what shapes we will use for our breeds:

```
to setup
    clear-all
    set-default-shape rocks "rocks-circle"
    set-default-shape papers "papers-circle"
    set-default-shape scissors "scissors-circle"
end
```

Note that the shape names must be in quotes, and must be written exactly as we wrote them when creating our custom shapes.

Next, we create our agents. In our previous work with NetLogo, we saw that the observer can create turtle agents with the `create-turtles` (or `create-<breeds>`) statement, and turtle agents can create other turtle agents with the hatch statement. In this case, we will use a third method: since we want exactly one turtle (whether of the rock, paper, or scissors breed) in each patch, we will simply tell each patch agent to sprout exactly one turtle on that patch:

```
to setup
    clear-all
    set-default-shape rocks "rocks-circle"
    set-default-shape papers "papers-circle"
    set-default-shape scissors "scissors-circle"
    ask patches [
        sprout 1 [

        ]
    ]
end
```

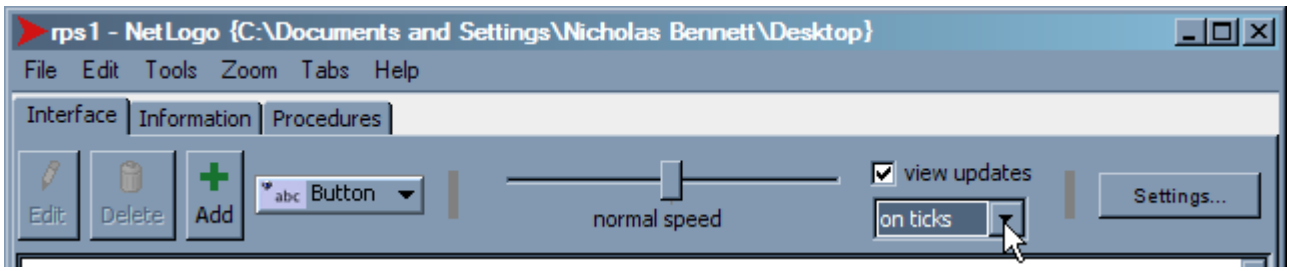
In the innermost set of brackets, we will write the code that each of the new turtles will execute as part of the setup procedure. Obviously, the most important thing that each turtle needs to do is select its strategy – which, in our model, is also its breed. We will do this randomly, using the `one-of` reporter. This reporter selects one of the items in a list at random. To use it in our model, we will create a list of the three breeds, and select from that list:

```
to setup
    clear-all
    set-default-shape rocks "rocks-circle"
    set-default-shape papers "papers-circle"
    set-default-shape scissors "scissors-circle"
    ask patches [
        sprout 1 [
            set breed one-of (list rocks papers scissors)
        ]
    ]
end
```

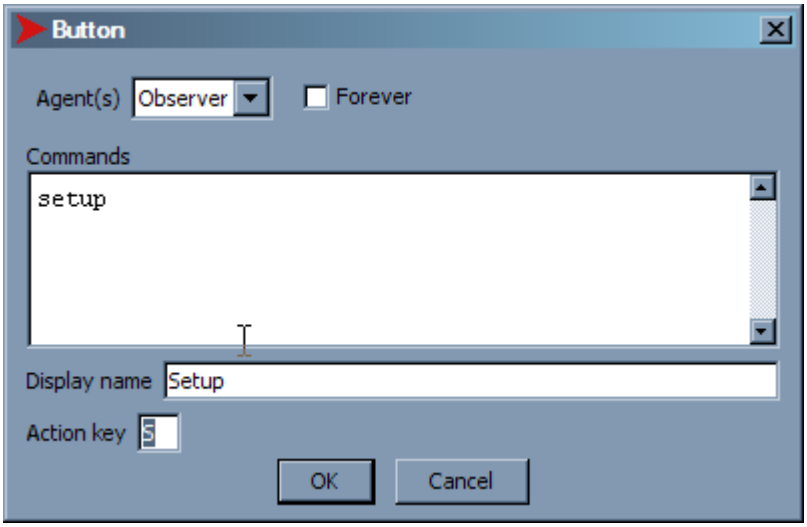
In this case, the parentheses around `list rock papers scissors` are essential; if they are omitted, NetLogo will report an error.

At this point, it's a good idea to check and save our code. (Remember to end the filename with .nlogo .)

Now, let's create a button that will run the setup procedure. But first, let's make a change that will make the model display update a bit more smoothly. Switch to the **Interface** tab, and look for the **view updates** checkbox (it will be close to the top of the window, just to the right of the speed slider). Make sure that the checkbox is checked, then pull down the menu below it, and select **on ticks**:



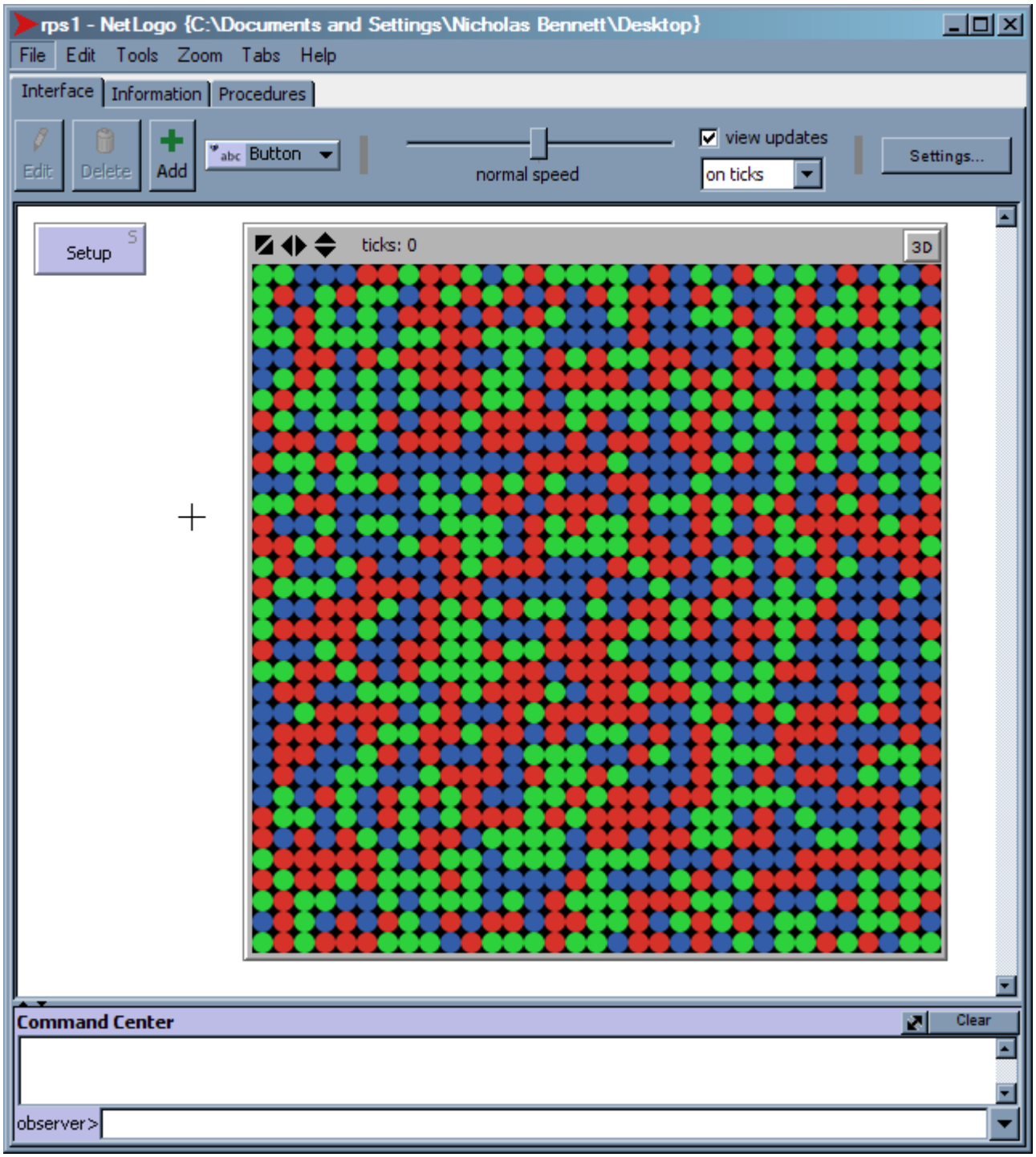
Now, for our setup button: right-click on the whitespace to the left of the NetLogo world , and fill in the Button dialog options:



(Note that the **Display name** and **Action key** values are not essential, and may be changed or omitted; the **Commands** are what is important here.) After specifying setup in the **Commands**, click **OK**.

Save your model again.

Now, let's see what it does so far: click your **Setup** button, and see what happens. You should see something like this:



Once you have fixed any problems that occurred, be sure to save again.

Teaching the Agents to Play Rock-Paper-Scissors

In each turn, each turtle agent will select one of its neighbors at random, and compare that neighbor's breed to its own breed. If the breeds are the same, the agent's turn ends in a tie. If the breeds are different, then the winner is determined by the normal rules of Rock-Paper-Scissors; however, the breed of the loser will then change to that of the winner.

Let's write a procedure called `play`, which will eventually implement the logic described above:

```
to play

end
```

First, we select the opponent at random. You might remember that there is a NetLogo command, `neighbors`, which gives us the set of the neighboring patches (i.e. those adjacent to the current patch). In this case, we will use a variation of that, `neighbors4`, which returns only those neighboring patches to the sides or up or down, but not those located diagonally. We will then use another command, `turtles-on`, to get the set of turtle agents that are standing on those patches. Finally, we will use `one-of` (which we have used before) to select one of those turtles at random.

```
to play
    let opponent one-of turtles-on neighbors4
end
```

Now that we know who our opponent is, we can compare breeds. We will do this comparison a few different times, for the different possibilities; to start, let's look for ties (if there is one, the turn is over):

```
to play
    let opponent one-of turtles-on neighbors4
    let my-breed breed
    let opponent-breed [breed] of opponent
    if (my-breed = opponent-breed) [
        stop
    ]
end
```

Now, let's identify the conditions under which the current agent wins the contest:

- The current agent's breed is `rocks`, and the opponent's is `scissors`.
- The current agent's breed is `papers`, and the opponent's is `rocks`.
- The current agent's breed is `scissors`, and the opponent's is `papers`.

Since we have already ruled out ties, we only need to check for the above combinations; if we don't find a match, that means the current agent loses the contest.

Pay close attention to the next bit of code; the parentheses and brackets are very important.

```
to play
    let opponent one-of turtles-on neighbors4
    let my-breed breed
    let opponent-breed [breed] of opponent
    if (my-breed = opponent-breed) [
        stop
    ]
    ifelse (my-breed = rocks and opponent-breed = scissors)
            or (my-breed = papers and opponent-breed = rocks)
            or (my-breed = scissors and opponent-breed = papers) [
        ask opponent [
            set breed my-breed
        ]
    ] [
        set breed opponent-breed
    ]
end
```

Check (and fix, as necessary) and save your code.

Telling the Agents to Play

The last procedure we need to right (for this basic version of the model) is one in which the observer tells all of the turtle agents to play. As we often do, we will call this procedure go :

```
to go
    ask turtles [
        play
    ]
    tick
end
```

Check and save the model.

Now, the contents of our **Procedures** tab should look something like this:

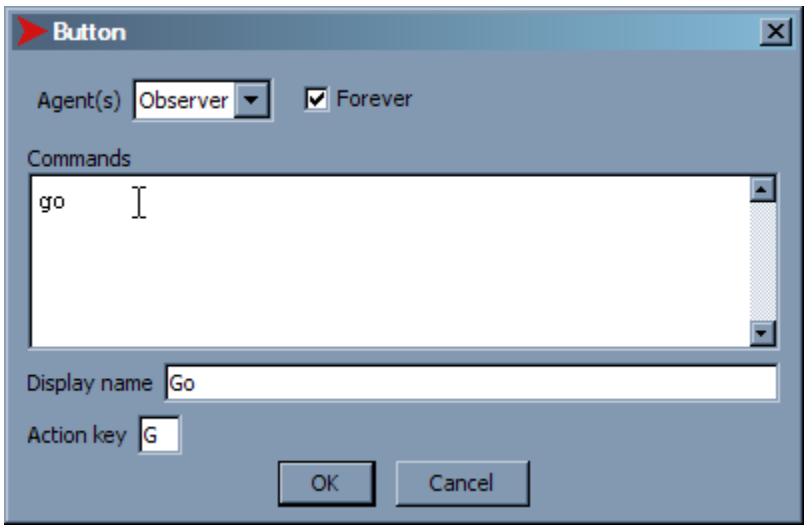
```
breed [ rocks ]
breed [ papers ]
breed [ scissors ]

to setup
    clear-all
    set-default-shape rocks "rocks-circle"
    set-default-shape papers "papers-circle"
    set-default-shape scissors "scissors-circle"
    ask patches [
        sprout 1 [
            set breed one-of (list rocks papers scissors)
        ]
    ]
end

to play
    let opponent one-of turtles-on neighbors4
    let my-breed breed
    let opponent-breed [breed] of opponent
    if (my-breed = opponent-breed) [
        stop
    ]
    ifelse (my-breed = rocks and opponent-breed = scissors)
            or (my-breed = papers and opponent-breed = rocks)
            or (my-breed = scissors and opponent-breed = papers) [
        ask opponent [
            set breed my-breed
        ]
    ] [
        set breed opponent-breed
    ]
end

to go
    ask turtles [
        play
    ]
    tick
end
```

Finally, let's make a **Go** button. Switch back to the **Interface** tab, and create a new button to call the `go` procedure. This time, make sure to check the **Forever** option:



Click OK. Save your model, and then click your new **Go** button to see what happens!

Things to Try

After getting a feel for how the model runs, use the **Settings...** button in the **Interface** tab to try one or more of the following changes:

- Change the topology from a torus to a rectangle, by unchecking the **World wraps horizontally** and **World wraps vertically** options.

- Make the world much smaller, by changing the **max-pxcor** and **max-pycor** values to 10 or 12.

- Make the world much larger, by changing **max-pxcor** and **max-pycor** to 50 (you will probably want to change the **Patch size** to 5, in order to see the entire NetLogo world on the screen).

Questions

1. If we view rock , paper , and scissors as three separate species, what sort of interspecies relationships are represented in the model?

    - Predation
    - Parasitism
    - Symbiosis
    - Competition
    - Cannibalism

2. Did you try any of the variations described on the previous page? If so, what differences (if any) in the model behavior did you notice?

3. In the description of the *E. coli* ecosystem, one of the experiments described included putting the *E. Coli* in a flask, and shaking it periodically, so that the individual bacteria didn't always remain in the same neighborhood . In our model, however, each turtle agent remains stationary, and only plays against its immediate neighbors. Can you think of any changes we might make in our model, to match the shaken flask *E. coli* experiment? (For now, don't worry about how we might program those changes; just speculate about what the logical changes should be.)