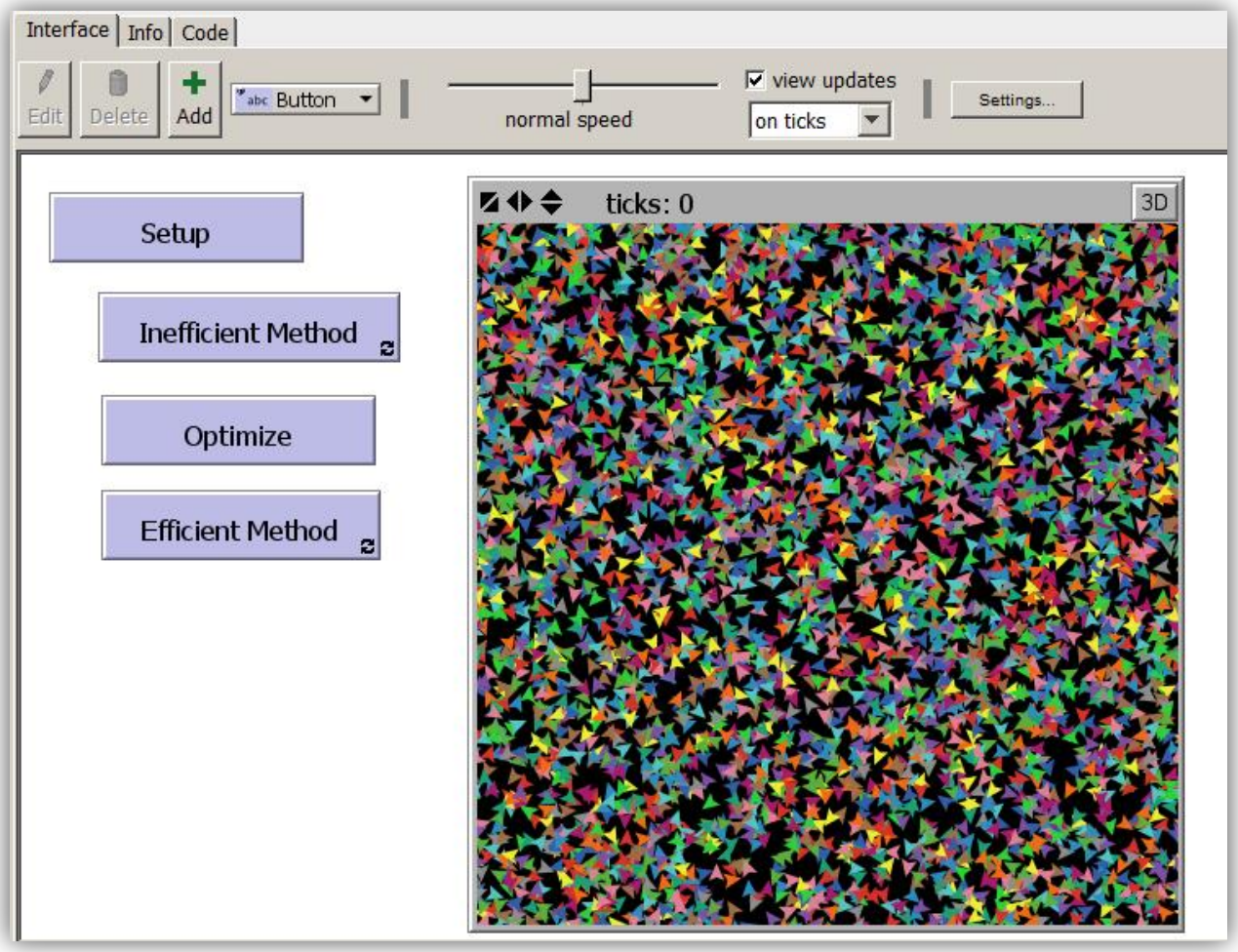
Lab 10: Color Sort



4000 Turtles – not yet sorted by color

Model Overview:

Color Sort must be a Netlogo model that creates 4000 turtles: each in a uniformly distributed, random location, with one of 14 uniformly distributed random colors, and each with a uniformly distributed random heading on a 33x33 patch torus world. Your goal is to teach each turtle to group up with all other turtles of the same color – using both an efficient and an inefficient method.

THE UNIVERSITY of NEW MEXICO

Setup:

The step is quite simple: all it does is clear the world (delete all turtles, clear all patches and reset ticks) create 4000 turtles, and set those turtles to random locations. By default, Netlogo creates each turtle with a uniformly distributed random heading and with one of 14 uniformly distributed random colors in random: thus, no extra programming. If you do not remember how to set turtles in random locations then review your Energizer Turtles Lab.

Inefficient Method:

The inefficient method is successful in getting all the turtles sorted by color. However, it does so rather slowly. The required algorithm for the inefficient method is:

a) Each turtle examines the other 3999 turtles to see which have a color the same as its own and puts all of these in a NetLogo ***agentset*** (a set of agents).
b) The turtle then randomly chooses one of the other turtles of the same color (in the created agentset).
c) The turtle turns to face the turtle chosen in step (b).
d) The turtle takes one patch size step forward.

Hint: How to do Inefficient Method Steps a and b:

The simplest way to do steps a and b of the inefficient method is to chain a few of Netlogo's reporters together: In NetLogo, **commands** and **reporters** tell agents what to do. A command is an action for an agent to carry out, resulting in some effect. A reporter is instructions for computing a value, which the agent then "reports" to whoever asked it.

one-of *agentset*	Given an agentset as input, **one-of** reports a random agent from that set. If the agentset is empty, the **one-of** reports **nobody**.
other *agentset*	Netlogo's **other** reporter is used to report a new agentset that is the same as the agentset it is given except with "this" turtle (the turtle in the current iteration of **ask turtles**) removed. The **other** reporter is used in this model because we want each turtle to move toward a turtle of the same color, BUT NOT to pick itself as the turtle to move toward.

turtles	Reports the agentset consisting of all turtles.
agentset **with** [*reporter*]	The **with** reporter takes two inputs: on the left, an agentset (usually "turtles" or "patches"). On the right, the reporter must be a boolean reporter. Given these inputs, **with** reports a new agentset containing only those agents that reported true -- in other words, the agents satisfying the given condition.

Putting this all together, we can build the powerful Netlogo statement:

```
let myTarget one-of other turtles with [color = mycolor]
```

The statement above reads from turtles in the middle to outer edges as:

turtles: Of the set of all turtles,

turtles with [color = mycolor]: Create an agentset of all turtles with **color** equal to **mycolor**. Note that **color** is the color of each turtle in the agentset and **mycolor** is a local variable that needs to be defined as the color of "this" turtle.

other: Remove "this" turtle (the turtle that is performing the action) from the agentset reported by **with [color = mycolor]**.

one-of: Report a random agent form the agentset reported by **other**.

let myTarget: assign the agent returned by **one-of**: to the local variable **myTarget**.

If you do not understand the words above, then read them again...and again. Then try them out. Do some experiments with this stuff. Just as the only way to learn to skateboard is to practice skateboarding, the only way to learn to program is to practice programming. If you say to your teacher "I do not understand, show me what to do", then, you will learn as much about programming as someone learns about skateboarding by watching others skate. Watching is an important part of learning, but it can only go so far. Ultimately, you need to try it – before you fully know how – and fall, get up and try again. Below are some practice grinds, kickflips and ollies:

First, 4000 turtles are too many to play with when testing.

Add an "ollie" button that clears everything and creates just 5 turtles that move a fixed distance from their starting location along their starting heading. Next, build an agentset of some of those turtles with a particular property. Finally, change some property of every agent in the agentset you build. Below is one example of this. You should try this out, then make a bunch of your own.

```
to Ollie
  clear-all
  reset-ticks
  create-turtles 5
  [
    pen-down
    forward 10
  ]

  let Lauren turtles with [pxcor > 0]
  ask Lauren
  [ set size 5
  ]
end
```

Note: the ollie button is not one of the graded requirements. It is a requirement only in that before understanding how to work with agentsets in simple examples and before gaining some practice in manipulating them, you have no hope of coming up with the code that is a graded requirement.

Optimize & Efficient Method:

These two procedures basically split up the tasks done in the inefficient method. To understand why this can get the program to run much faster requires some observations of the system:

1) Each turtle is given a color in setup and that color is not changed as the model runs.

2) Each turtle must take many steps before it all the turtles are sorted by color.

3) In the inefficient method, every tick, every turtle builds an agentset of all turtles that share its color, then picks a random element of that agentset to turn towards.

THE UNIVERSITY of NEW MEXICO

The key to efficiency is recognizing that while the random pick changes every tick and thus must be done every tick, each turtle's agentset of all turtles that share its color NEVER CHANGES! Therefore, each turtle's agentset of like colored turtles need only be built once. This can be done as follows:

1) Declare a new agent variable (in my implementation, I called it "**`agents_with_myColor`**") to store each turtle's agentset of like colored turtles.

2) Build the agentset in the **`Optimize`** procedure.

3) In **`Efficient_Method`**, pick a random member from the agentset.

When you get this to work, you will see that it saves lots of time. For one turtle to create its agentset of like colored turtles, that turtle must examine the color of all other turtles. Thus, the more turtles there are, the longer it takes for ONE turtle to build its agentset. Let n be the number of turtles (in this lab, n = 4000). Since EVERY turtle must build its own agentset, the total time to build an agentset is proportional to the time it takes one turtle to build its agentset, O(n) times the number of turtles that build lists, also O(n). The whole process then takes $O(n^2)$ computational time! In the inefficient method, this is done every tick.

Set verses List:

In Netlogo, there are things called ***sets*** and different things called ***lists***. An ***agentset*** is a set that contains only agents. In casual English, the words set and list are often used interchangeably. In computer science, however, these words have well defined and different meanings.

A ***set*** is an ***unordered collection*** where any ***repeated elements make no difference*** to the set. For example, the sets: {1, 2, 3, 4} and {4, 2, 1, 3} are equivalent. Also, if 2 is added to the set {1, 2, 3, 4}, then the resulting set is still {1, 2, 3, 4} since 2 was already an element of the set. With a set, it makes no sense to ask "what is the first element" since none of the elements have any particular order. Try this in Netlogo: create a list, add to it a repeated element, then show the list.

A ***list*** is an ***ordered collection*** where ***repeated elements DO make a difference***. Thus, the lists, [1, 2, 3, 4] and [4, 2, 1, 3], are different. If 2 is added to the list, [1, 2, 3, 4], it is important to ask ***where*** the 2 is added because the lists [2, 1, 2, 3, 4], [1, 2, 2, 3, 4], [1, 2, 3, 4, 2], ... are each different from one another.

THE UNIVERSITY of NEW MEXICO

Grading Rubric [20 points total]:

[A: 1 point]: Submit Netlogo source code named: `W10.`*firstname*`.`*lastname*`.nlogo`.

[B: 1 point]: The first few lines of your code tab are comments including your name, the date, your school, and the assignment name.

[C: 1 point]: The code in the code tab of your program is appropriately documented with "inline comments".

[D: 3 points]: Your program's "Setup" button works as specified.

[E: 4 points]: Your program's "Inefficient_method" button works as specified.

[F: 3 points]: Your program's "Optimize" button works as specified.

[G: 4 points]: Your program's "Efficient_method" button works as specified.

[H: 3 points]: Change your program's world settings by turning off the two world wraps checkboxes. Try to guess how you think this will affect the way the model runs? Run the model and see what happens.

In your program's Info tab, explain why the results are so different with world wrapping turned off.