MAKING COLORBLIND-FRIENDLY ABMS

Stan Rhodes

Accessibility Beyond the Defaults

In this appendix, we discuss how modelers can make the models they write in NetLogo more accessible to people with colorblindness.[1] We also challenge the common use of the default color schemes and scales in NetLogo by explicitly discussing their drawbacks for user experience, and offering alternatives. NetLogo made agent-based modeling accessible worldwide. However, we have to acknowledge that, at the time of this book, its accessibility has improved only modestly and its graphical options are limited. Graphics aren't everything, but they can help tell the story of the model visually. Although this appendix concentrates on making models colorblind-friendly, we hope that the considerations discussed here will help us all think a bit more about our graphical choices, and also what the future of agent-based modeling might—and should—hold.[2]

A Quick Primer on Colorblindness

Colorblindness affects people worldwide. However, "colorblindness" as a term is a bit misleading: most of those with colorblindness have a reduced ability to see differences in color, from just below normal color perception to complete inability to perceive the affected color.

The most common type—*deuteranomaly*—results in reduced perception of green light and difficulty distinguishing between red and green colors. Similarly, *protanomaly* is a reduced sensitivity to red light. Together, deficiencies in red and green cones are called red–green colorblindness. *Trianomaly* and *tritanopia* refer to deficiencies in

[1] You can find the full-color PDF version of this appendix in the ABMA Code Repo: https://github.com/SantaFeInstitute/ABMA/tree/master/appendix

[2] We assume modelers will consider overall visualization issues before, and in conjunction with, this appendix. For guidance on improving visualization for agent-based modeling, please see Kornhauser, Wilensky, and Rand (2009).

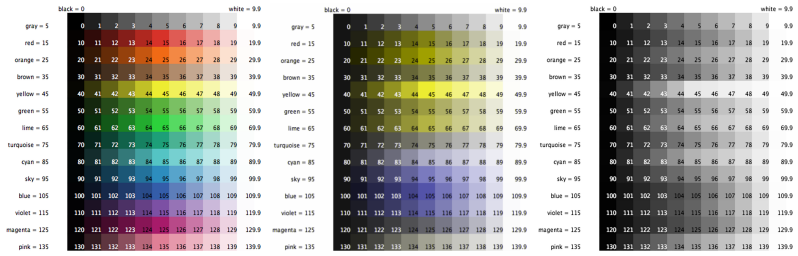

Figure A.0. The NetLogo palette in full-color, simulated deuteranopia, and simulated monochromacy.

perceiving blue. Monochromacy is the inability to distinguish among any colors.

The terminology used indicates the source of the deficiency: for example, deuter*anomaly* indicates a defect in the cones for green, whereas deuter*anopia* indicates the absence of cones for green.

Blue colorblindness occurs at roughly the same rate in males and females, while most people with red–green colorblindness are male. Monochromacy is much rarer than other forms of colorblindness in both males and females.

For a sense of how a person might with colorblindness might experience a graphic, see figure A.0, which simulates these difficulties in distinguishing colors.

Seeing the Problem

Color Oracle's grayscale option darkens all colors, including white, so use that option with some caution.

We recommend using a program to simulate colorblind views of your model. When adjusting the models in this book to be more colorblind-friendly, we used Color Oracle (available on Windows, Mac, and Linux) frequently.[3]

We recommend Cynthia Brewer's book, *Designing Better Maps: A Guide for GIS Users*. Brewer (2016)

A web search will find many pre-built colorblind-friendly color schemes. We recommend the ColorBrewer website (Harrower and Brewer 2003) for looking at existing color schemes for maps, which are often a good starting point in thinking about possible color schemes for your agent-based model.[4] Also, ColorBrewer and a colorblind simulator can help you develop intuition about why, for example, three distinct colors can work well, but six distinct colors lose distinguishability. After selecting different color schemes and numbers of total distinct data classes in ColorBrewer, use a colorblindness simulator to assess how it

[3] https://colororacle.org/

[4] https://colorbrewer2.org/

would appear to different types of colorblindness. Keep a healthy skepticism about colorblind-friendly palettes available on the web, as some work best for only one type of colorblindness, or have weak contrast in part of their scheme. Always do your own visual test of your model with a colorblindness simulator.

The Challenges of Agent-Based Modeling and Color Schemes

Making good color schemes for agent-based models can be hard because we are so constrained in our color options. Models offer many reasons to color everything differently: landscapes may have more than one value per patch; agents may be of different types, with different attributes. When looking at or developing a color scheme for your model, keep in mind that models with patches *and* agents will very likely need the agents to be set in colors distinct from the set of colors used for the patches, otherwise they won't show up well. You'll need to think about agent colors as one or more of the total colors in the overall color scheme.

Keep the Big Picture in Mind

It may be tempting to color everything distinctly, but think about what intervals and which distinctions are essential for the audience to understand the big picture. Consider where distinct colors are essential versus nice-to-have: for example, are intervals for land degradation really important, and if so, what are the largest viable intervals for those values? Difficulty in simplifying and emphasizing parts of the model may reflect areas where model visualization should be improved.

See Kornhauser, Wilensky, and Rand (2009) for guidance and a process to look at the big picture and refine the visualizations of your model.

Shapes Can Help Reduce the Need for Distinct Colors

Shapes expand your ability to distinguish between patches and agents. Do different agents really need different colors, or could you use different shapes instead? A dark triangle on a patch may denote mountainous terrain as well or better than a distinct color. Different or smaller shapes may differentiate juvenile animals from adults. With enough contrast between colors, a smaller shape can be placed on a larger shape. For example, a light-colored spot can be placed on an animal shape to indicate disease, or on a plant shape to indicate flowering or fruiting. These

This appendix includes a section on shapes near the end.

smaller shapes can use colors that are the same or similar to another light color in your palette. The shape and contrast make the distinction, so the color similarity doesn't matter; no one will confuse a yellow dot on a deer for a yellow patch denoting sandy soil.

Which Colors Work, Which Don't

You may be asking "Which colors should I use, then?" Unfortunately the answer is, "It depends." It depends on the model, and it depends on what distinctions you want users to be able to make when viewing the model. We'd recommend using a few heuristics to guide your process.

CONSIDER WHICH GRADIENTS ARE CRUCIAL

Consider any crucial gradient(s) you want to display first, as it will be the biggest constraint on your remaining colors. What color makes the most sense for the maximum value of the gradient? What are the largest viable intervals for that value, and thus the fewest intervals that will work to show differences? Would a lighter or darker gradient make sense, and would it be possible to only use a light or dark gradient, rather than the whole range? Consider the full range of the gradient you want to display and then how many shades within the gradient are important to include. Fewer shades with larger intervals may be sufficient to show the differences you need.

IF GREEN IS ESSENTIAL, START WITH IT

Consider if green is a crucial color in your model, because that will constrain your options. This really holds true for any crucial color. However, green is special: green is used in agent-based models all the time as an indicator of vegetation, yet it's also a problematic color for most types of colorblindness. Rethinking the use of plain, pure green in models is a core challenge of making them colorblind friendly, and often a good starting point for considering the palette of the model. Would lighter greens or darker greens be a better choice? Could you keep the meaning of the green while changing the hue to a bluer or yellower green?

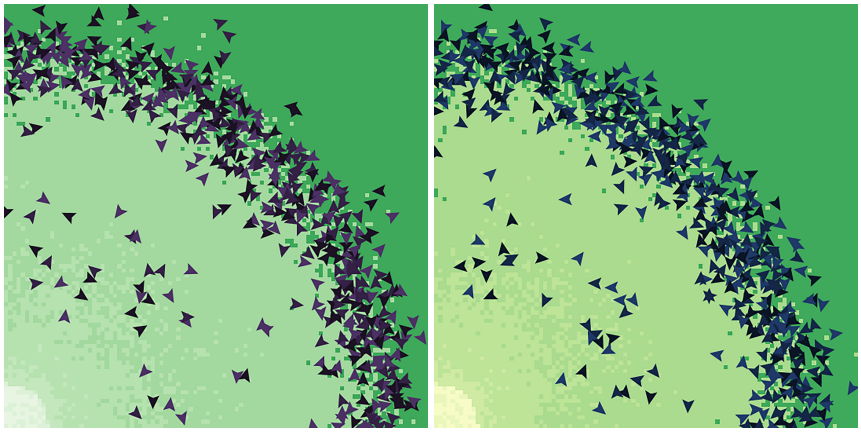

Figure A.1. If a green gradient is essential for the model, the best option for two contrasting color gradients is light green and dark violet (left). Yellow-green and dark blue (right) is a second, but much less ideal, option.

SPLIT THE COLOR SCHEME INTO LIGHT AND DARK OPTIONS

If having many colors seems unavoidable, assemble your color scheme options by dividing it up into two parts: lighter and darker. These may be gradients or discrete sets. If only one gradient is needed, light agents can be used on the dark gradient, or vice versa. If two gradients are needed, consider using only black and/or white agents.

Try to avoid brown if you can. Being a combination of red and green, it's usually not worth the trouble unless it really is essential to visually communicate an aspect of the model.

Gradients in colors schemes

If using a green gradient, the best option for two contrasting color gradients is light green and dark violet. See figure A.1. Yellow-green and dark blue can be used, but is not ideal for tritanopia, as the hues look similar.

In this example, we apply the NetLogo named color to the turtles and we use the palette extension to apply the "Greens" color map from ColorBrewer to the patches.

We will cover use of the palette extension in more detail later in the appendix.

CODE BLOCK A.0

```
create-turtles 500 [
  set size 4
  set color (random 3 + violet - 4)
]
```

CODE BLOCK A.0 (cont.)

```
ask patches [
  set pcolor palette:scale-gradient
    palette:scheme-colors
    "Sequential" "Greens" 5 soil_quality 0 100
]
```

Discrete values in color schemes

Discrete values need to be farther apart in contrast than colors in a gradient, which means that while four discrete values can be colorblind friendly, five or more discrete values will become indistinguishable for people with colorblindness. You can use ColorBrewer to find discrete values that will work if you use their given RGB values with NetLogo's `set color` and `set pcolor`. ColorBrewer does not have colorblind-friendly discrete sets above four discrete values.

The use of any gradient or background colors constrains your available colors. Although in the model's VIEW, the patches' colors are the likely constraint, NetLogo's white-background plots are another common constraint. We'll provide a few options here for discrete value sets that are distinguishable on plots. If the colors aren't to be used on plots, you have slightly more options.

If you're plotting discrete colors, you'll need to avoid light colors like yellow so that the lines are visible on the plot. Five discrete colors inevitably results in one being light-colored.

▷ *Discrete values using NetLogo's named colors*

- Four-color discrete set:

  `red`, `orange + 2`, `blue`, `black`.

- Five-color discrete set (see fig. A.2):

  `red`, `orange + 2`, `blue`, `black`, `lime + 3`.

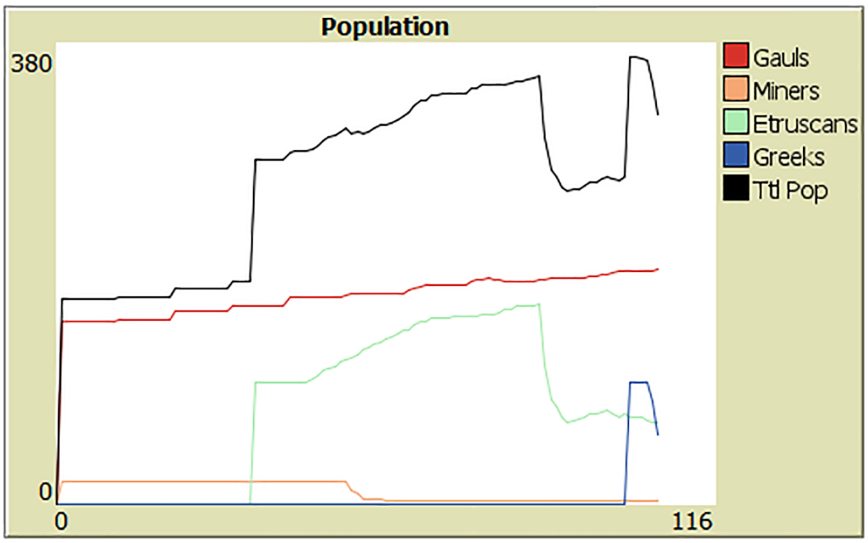

Figure A.2. This set of five discrete values can be used in a plot, but adding a further color will make two of the plot colors indistinguishable. Ideally, we would use only four.

▷ *Discrete values using RGB values from viridis*

- Five-color discrete set:

CODE BLOCK A.1

```
[252 228 30]
[144 214 45]
[31 146 115]
[42 72 122]
[54 0 65]
```

NEEDING MANY COLORS SUGGESTS HIGH MODEL COMPLEXITY

If you find yourself needing a lot of colors to distinguish between elements and components of the model, that suggests the model may be too complex to be easily understood by others. In that case, using a lot of colors is unlikely to improve the comprehensibility of the model, and you should consider simplifying what you're communicating about the model. You may be able to divide your visualization into separate concepts and find a reduced color option for each of them.

In cases where just one or two more colors seem needed to distinguish elements of the model, take another look at whether using shapes

Figure A.3. Setting the color bounds to match the bounds of the the variable range often leads to a confusing mess.

might help. If not, or if you are using a lot of shapes already, reassess the complexity of the model in light of what you want to communicate.

The Challenges of Default NetLogo Color Gradients

NetLogo's default color scales present some challenges for user interpretations of models, which then make colorblind-friendly considerations a little extra tricky. Consider a model that has patches with a soil fertility score between 0 and 100. We might normally map that to a color using `scale-color`:

CODE BLOCK A.2

```
ask patches [ set pcolor scale-color
    green fertility 0 100 ]
```

The result is low-fertility patches that are nearly black, and high-fertility patches that are nearly white.

Visually, these defaults cause patches to blend with agents, and conflict with a basic intuition that fertile soil produces very green grass (instead, here the highest fertility is depicted as white). One way to adapt NetLogo's default behavior for an improved visual gradient is to set the range limits to be outside the fertility variable's range. We can use this for

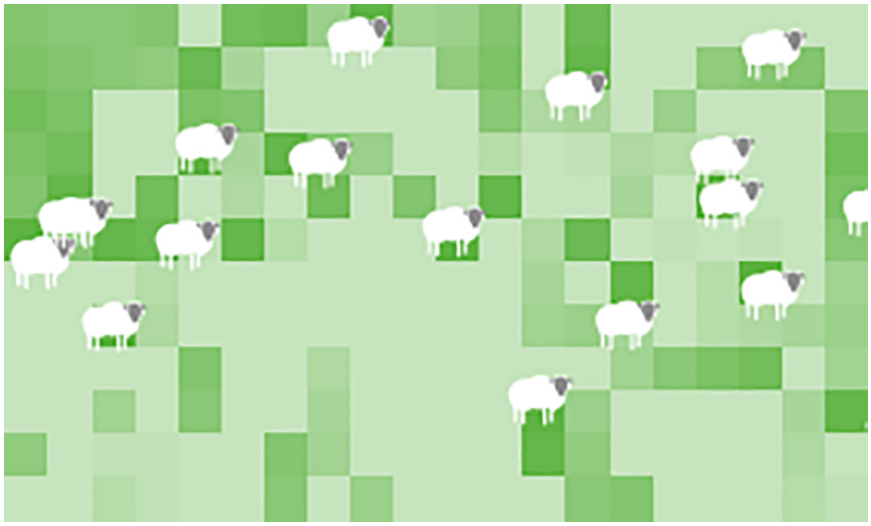

Figure A.4. Setting the color bounds out of range enables more intuitive color gradients than matching the bounds with the variable range.

both light and dark gradients because we can invert the minimum and maximum values to reverse the color gradient. For example, we might want low fertility to be light green and high fertility to be dark green. Before we increase the range limits, we place the larger number first to invert the scaling so that low values are lighter and high values are darker (`100 0` instead of `0 100`). Then, with the color gradient now going from light to darker, we extend the range by adding some padding (e.g., ~20) to the resulting numbers so that the extremes are outside the variable's range:

CODE BLOCK A.3

```
ask patches [ set pcolor scale-color
    green fertility 120 -20 ]
```

Guessing at how much padding to add is, well, guessing. If we could normalize the range between 0 and the max of a variable for any variable, we could avoid NetLogo's color variant extremes consistently without guesswork. And we can!

AVOIDING EXTREME LIGHT AND DARK COLORS IN VANILLA NETLOGO

Here we provide a more programmatic and consistent way of setting the appropriate outside-the-variable-range bounds to avoid light and dark

extremes, which also uses a vivid color (one of NetLogo's named colors) at one end of the gradient instead. In figure A.5 we show four gradients: vivid-to-light, light-to-vivid, vivid-to-dark, and dark-to-vivid. The code for each requires a variable, we use `fertility`, that ranges from 0 to a maximum value. We also set `varmax` to be the maximum value of `fertility`, which should be set in the code (e.g., `set varmax 100`). Here is the code for each in turn:

▷ Vividest color = 0, light color = variable's max value:

CODE BLOCK A.4

```
let upperbound (varmax + varmax / 2)
let lowerbound (-1 * (varmax + varmax / 2))
set pcolor scale-color green
    fertility lowerbound upperbound
```

▷ Light color = 0, vividest color = variable's max value:

CODE BLOCK A.5

```
let upperbound (2 * varmax + varmax / 2)
let lowerbound -1 * (varmax / 2)
set pcolor scale-color green
    fertility upperbound lowerbound
```

▷ Vividest color = 0, dark color = variable's max value:

CODE BLOCK A.6

```
let upperbound (-1 * (varmax + varmax / 2))
let lowerbound (varmax + varmax / 2)
set pcolor scale-color green
    fertility lowerbound upperbound
```

▷ Dark color = 0, vividest color = variable's max value:

CODE BLOCK A.7

```
let upperbound -1 * (varmax / 2)
let lowerbound (2 * varmax + varmax / 2)
set pcolor scale-color green
    fertility upperbound lowerbound
```

These are an improvement over the default behavior, and are color-blind friendlier. Often they will do the trick if you need to use a NetLogo word color; you can use any of NetLogo's named colors in place

Figure A.5. Programmatically setting the color bounds out of range enables fine, consistent control of the color gradient with any range of variable values. From left to right: vivid-to-light, light-to-vivid, vivid-to-dark, dark-to-vivid.

of `green`. However, we can usually do as well or better with the palette extension because it can use the predefined, and multi-color, ColorBrewer palettes.

USING THE COLOR-PALETTE EXTENSION

The palette extension is a default extension included in NetLogo, and its functions can be made available within a NetLogo by including `extensions [palette]` at the start of the code. Here is an example of using the palette extension to set a color gradient:

CODE BLOCK A.8

```
ask patches [
  set pcolor palette:scale-gradient
    palette:scheme-colors
    "Sequential" "Reds" 3 artifact_density 0 100
]
```

The NetLogo palette extension doesn't have the sequential ColorBrewer class "YlGn" (which is a really useful one!), but we provide an implementation of it below.

The easiest way to use the color-palette extension is with ColorBrewer settings. Make sure to test the qualitative ColorBrewer schemes in your model with a colorblindness simulator tool. Some of the schemes work better than others, especially on plots.[5]

[5] The original documentation on the palette extension is a bit scattered; please see:
https://ccl.northwestern.edu/netlogo/docs/palette.html
http://ccl.northwestern.edu/papers/ABMVisualizationGuidelines/palette/
http://ccl.northwestern.edu/papers/ABMVisualizationGuidelines/palette/doc/NetLogo%20Color%20Howto%202.htm

Advanced Palette Extension: Making Custom Palettes

The palette extension also gives us advanced options for colors where we can specify the entire scheme with RGB values. In our experience, this is often not worth the effort, and also makes for very hefty code. Use it sparingly and only when you can't find another viable alternative.

Currently the YlGn palette from ColorBrewer has not been implemented in NetLogo's palette extension, so we will implement it here as an example using ColorBrewer's YlGn color scheme with five data classes. To get these values, we went to the ColorBrewer website[6] and selected the YlGn sequential scheme, then made sure we copied the RGB (versus HEX or CYMK) values. The copied text will require some cleanup to make the RGB values the format NetLogo requires.

The nine-data class version would have nine RGB colors in the list.

CODE BLOCK A.9

```
ask patches [
  set pcolor palette:scale-gradient
        [
          [255 255 204]
          [194 230 153]
          [120 198 121]
          [49 163 84]
          [0 104 55]
        ]
        variable-name 0 100
  ]
```

We recommend using a text editor with good find-and-replace functionality when working with large color palettes to clean up commas and insert the square brackets for NetLogo-bound code. This includes programs such as Notepad++ (Windows) or Atom (Windows, Mac, Linux).

Once you assign colors in an RGB format (e.g., `255 255 204`), the typical NetLogo approach of adding or subtracting values (`set pcolor pcolor +3`) to lighten or darken a color will no longer work. The color's data structure is now different; RGB is a list of three 0-to-255 values rather than single number from 0 to 140. Adjusting color in this way is bad coding practice as well; updating the color *only* would mean it no longer represented the underlying value. When the color is set from an underlying variable initially, every subsequent color change should use that underlying value to set the color.

[6] https://colorbrewer2.org/

You might wonder if you can increment RGB values in the same way you would increment single-value or named colors in NetLogo. Unfortunately, changing RGB values isn't a simple linear process, so adding or subtracting values from one of the three RGB values (or all of them) won't give consistent results.

CAN WE USE SOME OPTIMAL COLOR SCHEME?

If we were considering gradients for data visualization only, in most cases we would want to use a color scheme that was uniform to all users, but which also maximized the perceptual range of all users (colorblind or not). A number of color maps fit these criteria well enough, including viridis, magma, inferno, and plasma (created by Stéfan van der Walt and Nathaniel Smith) and cividis (created by Jamie R. Nuñez, Christopher R. Anderton, and Ryan S. Renslow). Unfortunately, implementing these color maps in NetLogo is no easy task: you'll need to convert the map to a long list of RGB values for the palette extension (e.g. code block A.9).

In some cases, these color maps will need to be found inside a package. For example, with viridis, you can load the package in R and look at the color map as a dataframe. The values in the dataframe can then be parsed into RGBs and written to a string that can be used in NetLogo.

An RMarkdown document for creating a NetLogo list from the viridis colors is available in the ABMA Code Repo as *viridis_color_calcs.rmd*.

Keep in mind that agent-based models are usually more complicated than a single gradient, which means that as more colors are needed for agents, fewer colors are available for gradients. The gradients, then, will need to be pared down to just a few colors; you will need to pick a subrange in the color map that works well for your other color choices.

Working with Shapes

Shapes offer a lot of possibilities for improving your model visually, both for distinguishing different types of agents and for clarifying landscape or environmental features.

OUTLINING SHAPES FOR BETTER VISIBILITY

Generally, lighter-colored landscapes with vivid-colored agents work well. However, sometimes a landscape may make agents a little harder to see. Heterogeneous landscapes using color gradients can cause visual

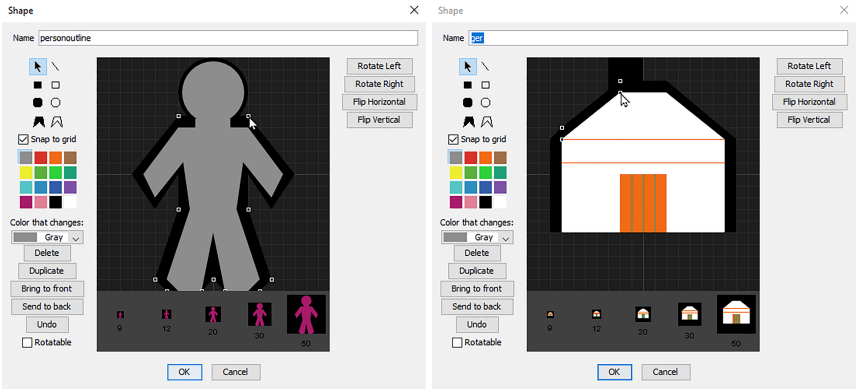

Figure A.6. The shapes editor can be used to outline existing shapes in black for higher contrast and for avoiding agent colors "bleeding" into patch colors.

confusion when part of the gradient has a similar contrast to the agent. In these situations, the agents need more visual "pop."

A black outline should give agents the necessary contrast to work on any landscape. Unfortunately, most default shapes have no black outline; fortunately, adding a black outline to most shapes is straightforward, see figure A.6. NetLogo comes with a shapes editor built in, accessible via TOOLS > TURTLE SHAPES EDITOR.

Remember that you can import shapes from the shape library or import them from another NetLogo model. See https://ccl.northwestern.edu/netlogo/docs/shapes.html

USING SHAPES FOR STATES OR TERRAIN

Shapes may better communicate a terrain type than patch color alone: for example, a triangle may better represent a mountain than trying to pick the "right" mountain color, as in figure A.7.

Shapes may also help denote landscape states better than color alone, particularly when the landscape has events such as snow, rain, seeds germinating, or plants fruiting. See figure A.7.

WHEN USING SHAPES, USER EXPERIENCE IS STILL KING

How many shapes are too many? Again, it depends entirely on the model. A good general approach is to modify one feature at a time, run the model, and ask yourself whether the visualization feels busy. Sometimes taking a little time off from the model before revisiting it can help you see it afresh. If you're still not sure, show it to someone for feedback.

Kornhauser, Wilensky, and Rand (2009) discuss visual design principles and considerations in great depth.

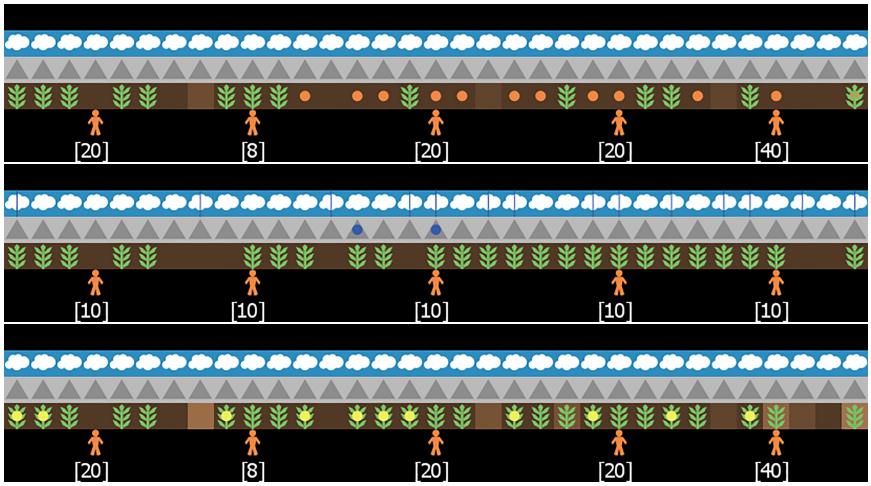

Figure A.7. Shapes can tell part of the story when the paired colors provide sufficient contrast. Top: Seeds, as circles, sprout into plants. Middle: Lines in the clouds represent rain, and water running down to plants are dots on the mountains. Bottom: Plants with mature ears of maize have yellow dots with high contrast.

As Accessibility Evolves, So Does Our Thinking

Keep in mind that although it would be nice to "do it right the first time," that is an unrealistic expectation for any modeler. Developing a clear, concise, and accessible model is an iterative process. Making models colorblind-friendly involves trade-offs in meaningful color-based communication, but we have found that understanding these trade-offs and working within these constraints has improved the depth of our understanding of the use of color, and our approach to modeling overall. We hope you will find the same benefits!