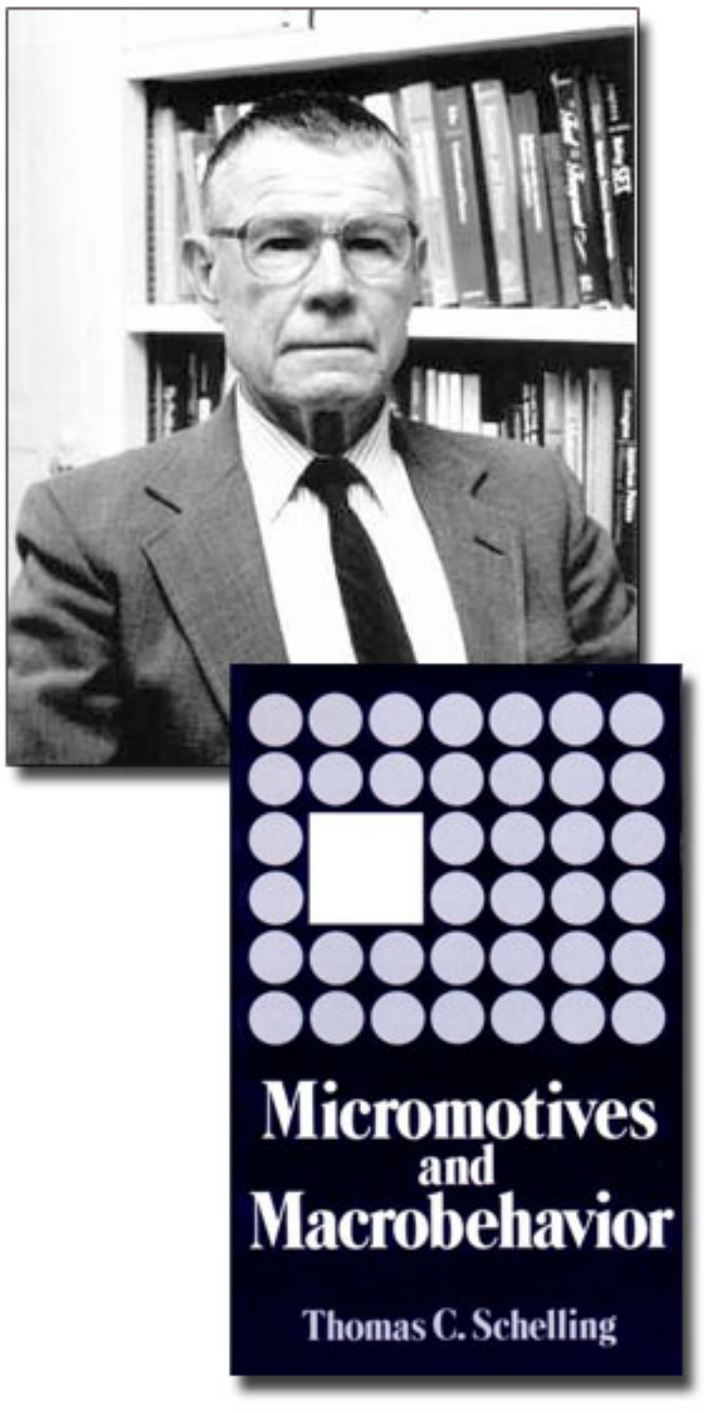
Introduction to Agent-Based Modeling in Netlogo

Aaron L Bramson

If you haven't already, download and install Netlogo from http://ccl.northwestern.edu/netlogo/download.shtml

Then go to http://tinyurl.com/nkn4n4 to download the SegregationEnhanced.nlogo file we'll be working with.



Thomas C. Schelling

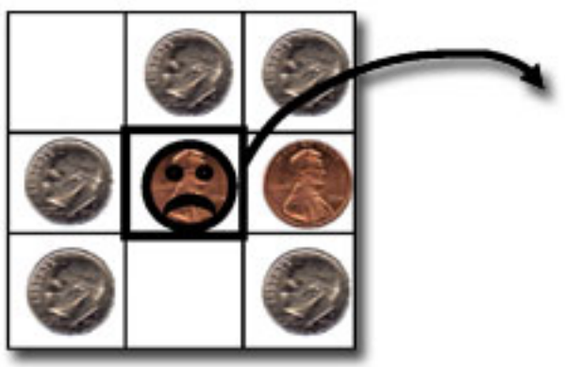
- Father of Agent-Based Modeling
- 2005 Nobel Prize Winner in Economics for work in Game Theory
- Clearly explained and motivated bottoms-ups approach in his 1978 book "Micromotives and Macrobehavior"
- Famous Segregation (aka "Tipping") model is the first modern agent-based model

Segregation Model

- There are two types of agents and they have a percentage of neighboring agents that they want to be of the same type.

- The neighbors are agents in any of the eight surrounding spaces of the grid.

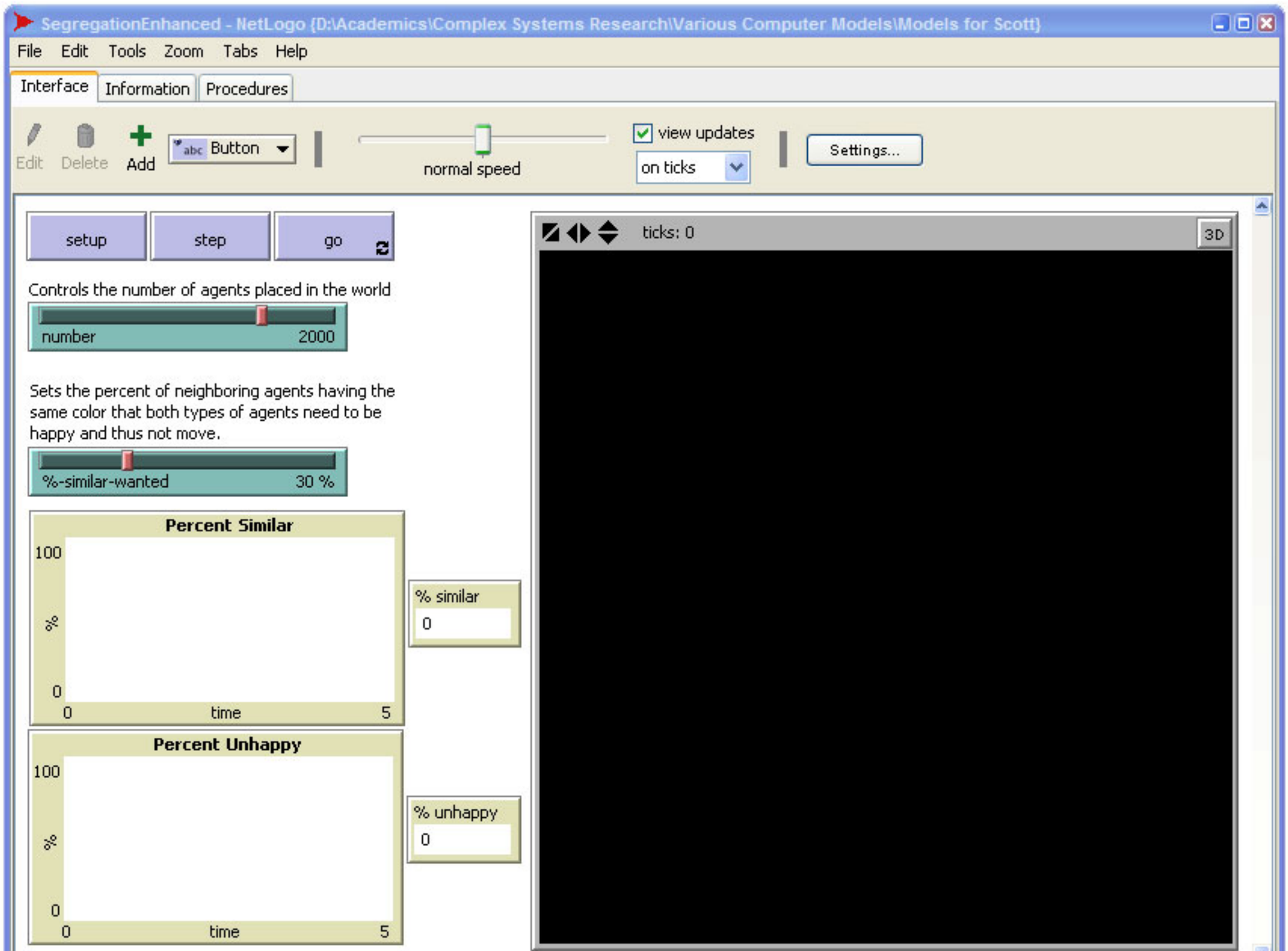
- Unhappy agents move around to empty spaces until they are happy.



- Once all the agents are happy the model stops running.

Introduction to Agent-Based Modeling in Netlogo

Aaron L Bramson



Aaron L Bramson

Elements of the Interface

BUTTONS

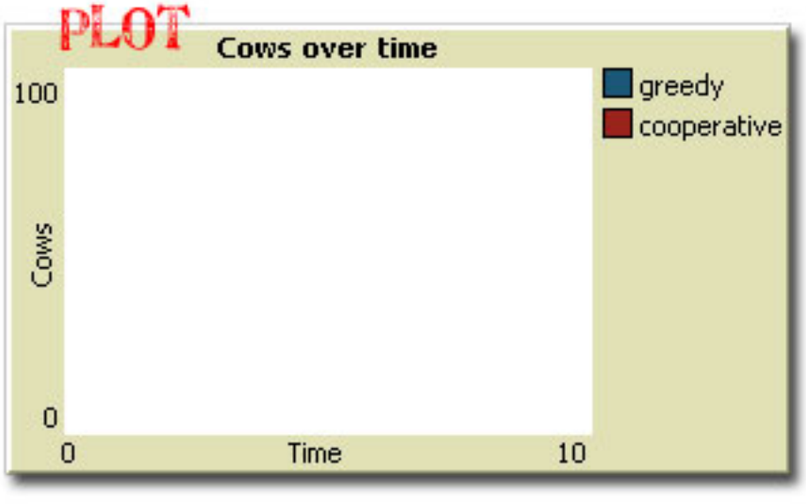
ONCE FOREVER

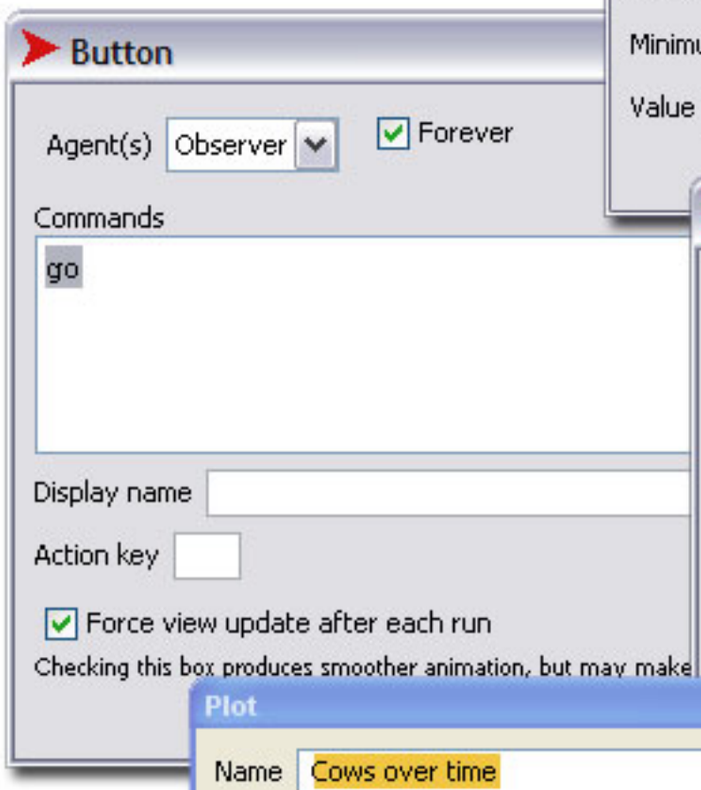
SLIDER

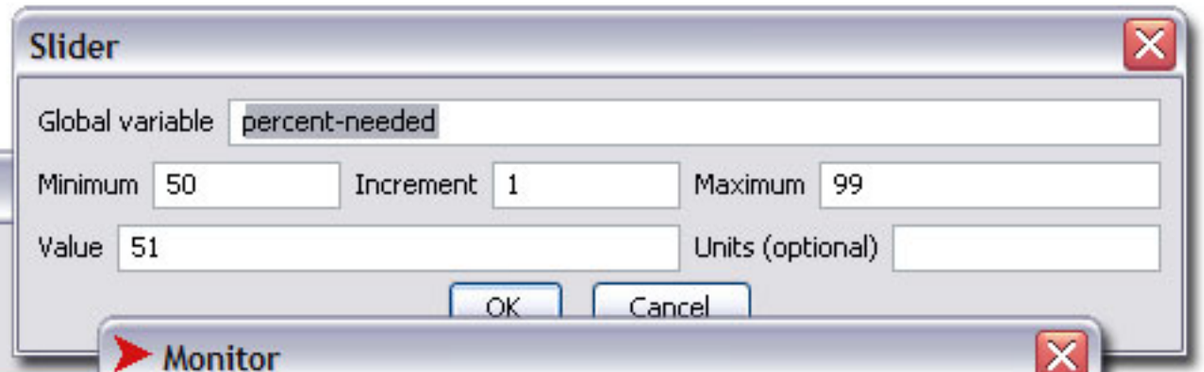
SWITCH

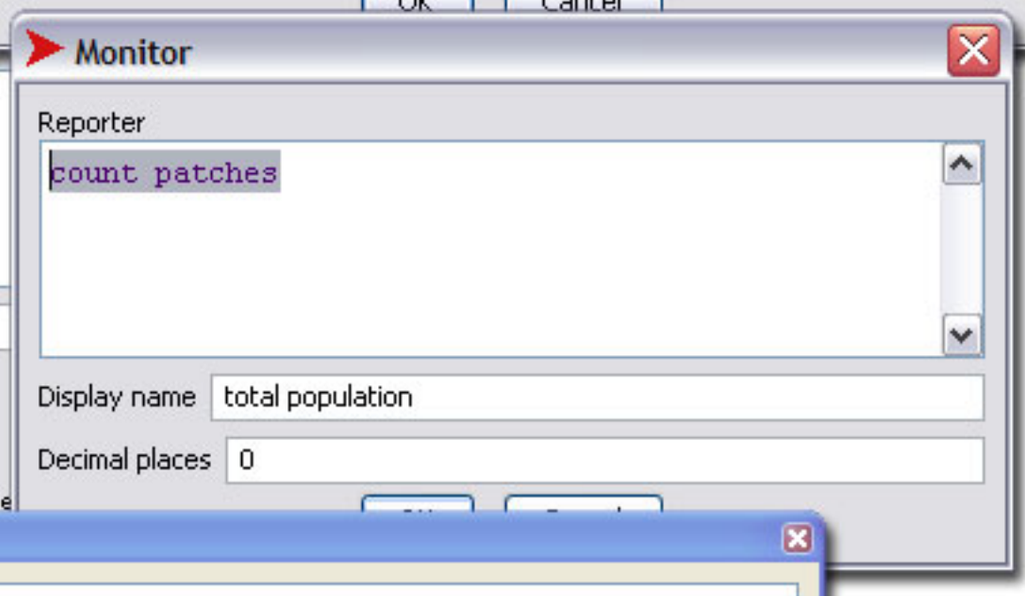
MONITOR

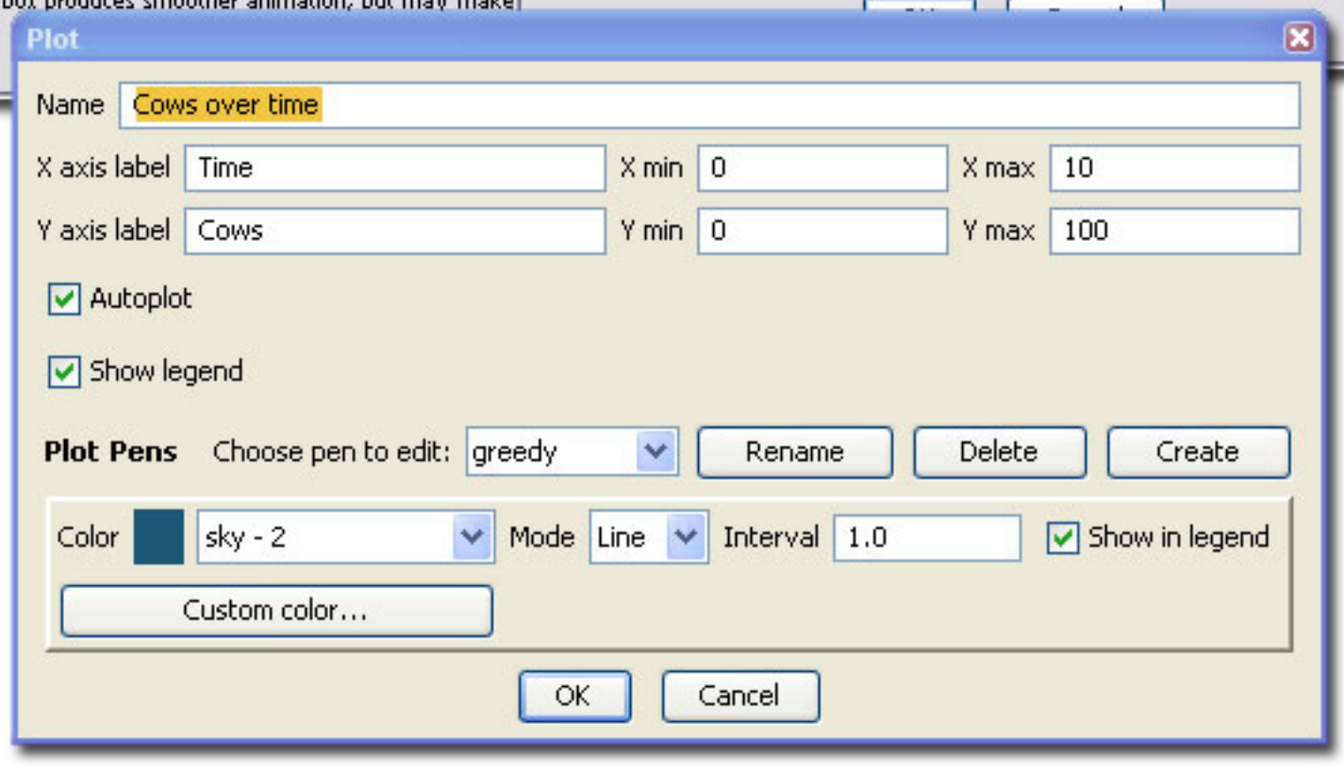
PLOT



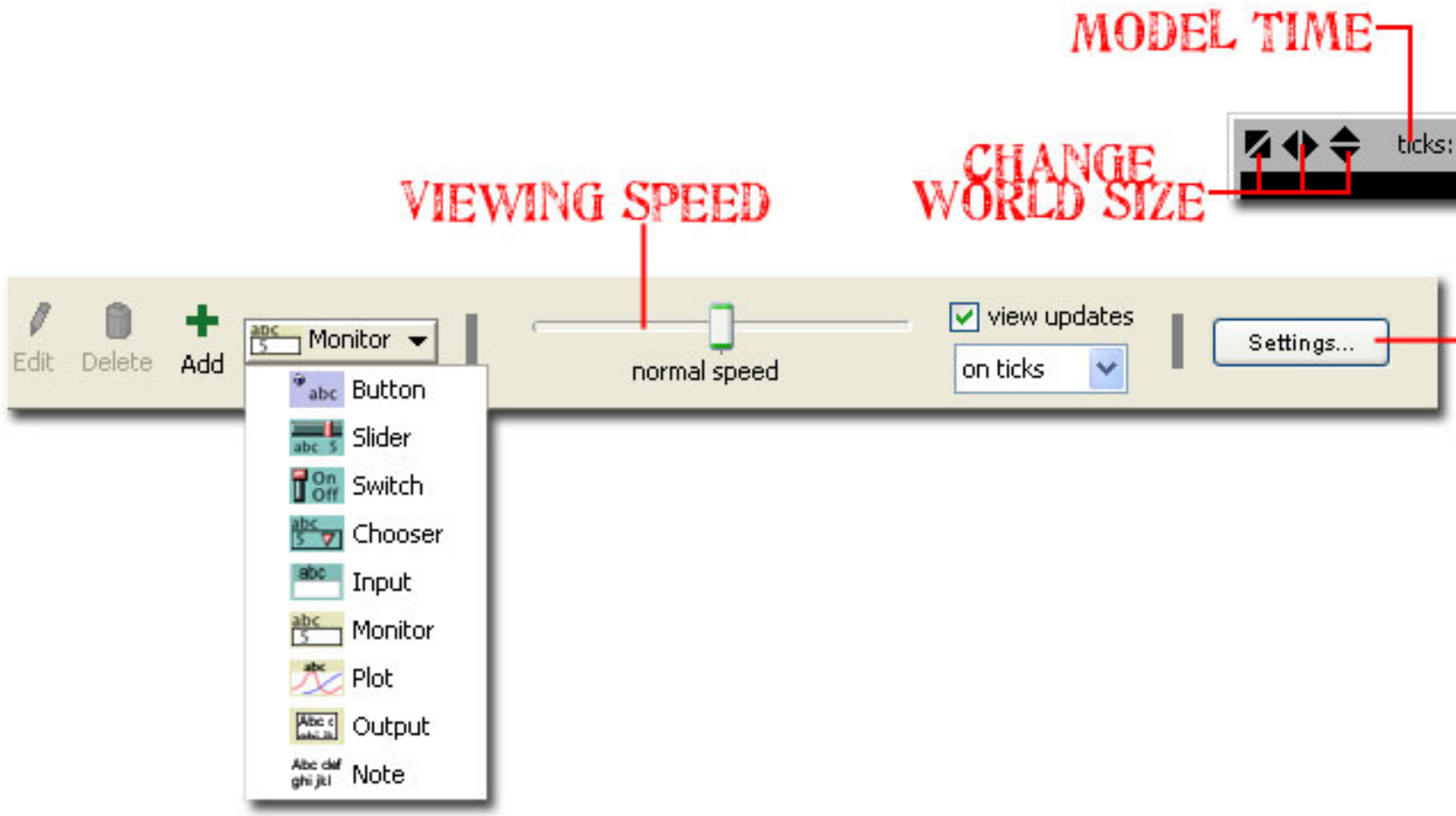




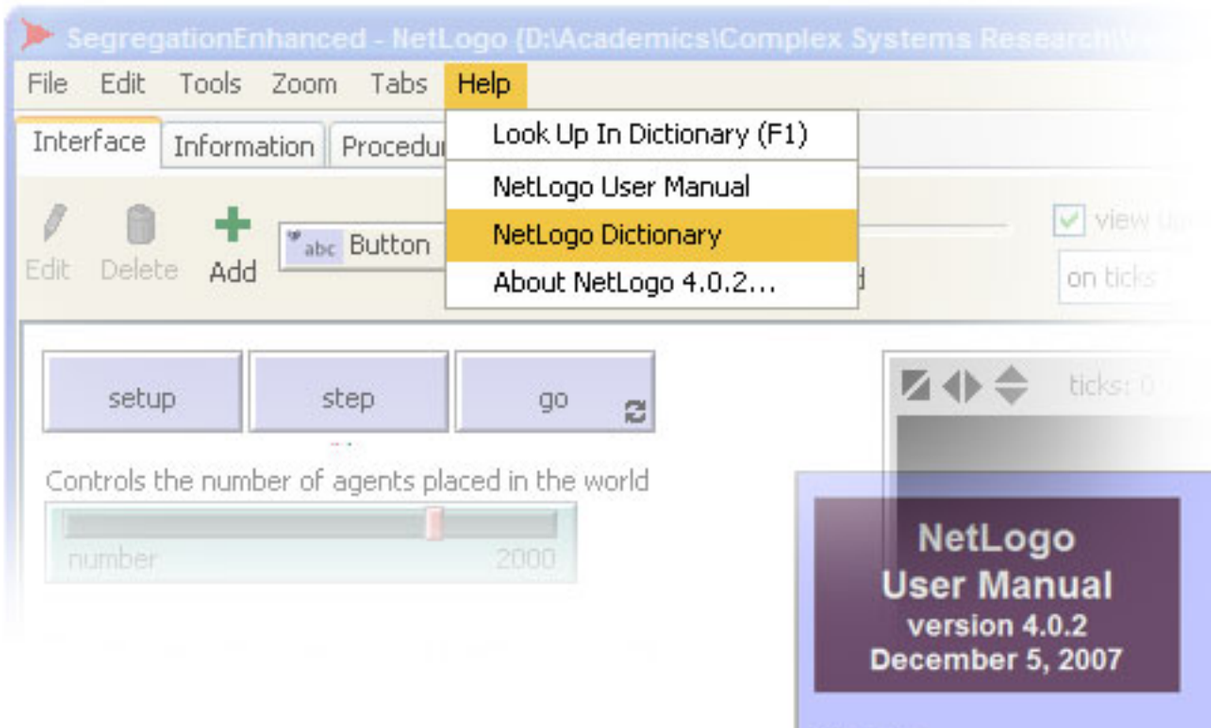




Elements of the Interface



There's more,
but that's enough to get going.



The dictionary has a description of every primitive and many examples.

Also see the Guides for more in-depth explanations of certain features.

Using the Model

- Click the SETUP button to set up the agents.
- There are equal numbers of red and green agents.
- Click GO to start the simulation.
- If agents don't have enough same-color neighbors, they jump to a nearby patch.
- The NUMBER slider controls the total number of agents.
- The %-SIMILAR-WANTED slider controls the percentage of same-color agents that each agent wants among its Moore neighbors.

Aaron L Bramson

Let's Look at the Code

```
(number / 2) turtles
  [ set color green ]
;; these commands call procedures written elsewhere in the code (scroll down to see them).
update-variables
do-plots
end

;; this method is run through each tick that the model goes through.
to go
  if all? turtles [happy?] [ stop ]
  move-unhappy-turtles
  update-variables
  tick
  do-plots
end

;; if all the turtles are happy then the model breaks out of the go loop.
;; if NOT all the turtles are happy, then the OBSERVER calls this method.
;; this advances the time counter by one.
;; this is the end of the go method,
;; but if a forever button is used it repeats from the beginning.

;; the OBSERVER calls this method in the go method,
;; and it then asks turtles with a particular property to do something.
to move-unhappy-turtles
  ask turtles with [ not happy? ]
    [ find-new-spot ]
end

;; this method has the turtles find a new empty spot by trial and error rather than informed search
to find-new-spot
  rt random-float 360          ;; turns to face a new random direction
  fd random-float 10           ;; moves forward between 0.0 and 9.9999... patches
  if any? other turtles-here   ;; keep going until we find an unoccupied patch
    [ find-new-spot ]
  move-to patch-here           ;; move to center of patch
end

;; the updating is split into two sub-methods so that they can be called seperately
;; and used seperately in different applications without need to be rewritten.
to update-variables
```

Everything Clear?

Extensions of the Model

1) To change the model to Schelling's "Surfers and Swimmers" model add another slider for %-similar-wanted.
   Make one for red agents and the other for green agents so they can have different tolerance levels.

2) Add another type of agent and assign it a new color.
   Change the setup procedure so that there are equal numbers of each type of agent.

3) Change the agents' shape so that it's easier to see the distribution of agents.
   Go to the tools menu and select the Shapes Editor and then choose the shape that makes sense to you.

4) Add sliders for the number of each type of agent so you can have an uneven distribution of agent types.
   Run some experiments on different mixes and note any qualitative changes that result.

5) Add a mutation rate so that unhappy agents have a percentage of changing their type rather than moving.
   Note that changing thier type may not make them happy depending on the percentage required.
   Does this speed up or slow down the time to universal happiness?

6) Change the mutation rule into a learning rule so that agents copy the type of turtle that has the highest percentage of happiness among its neighbors.
   That is, find which type of agent among the neighboring agents has the highest percentage of happy agents and copy that type.

7) Instead of just the Moore neighborhood, add a radius slider and have agents calculate happiness based on all the agents within that radius.
   Does this make it easier are harder to be happy?
   What effects does this have on the distribution of agents?

Extension 1: Surfers and Swimmers

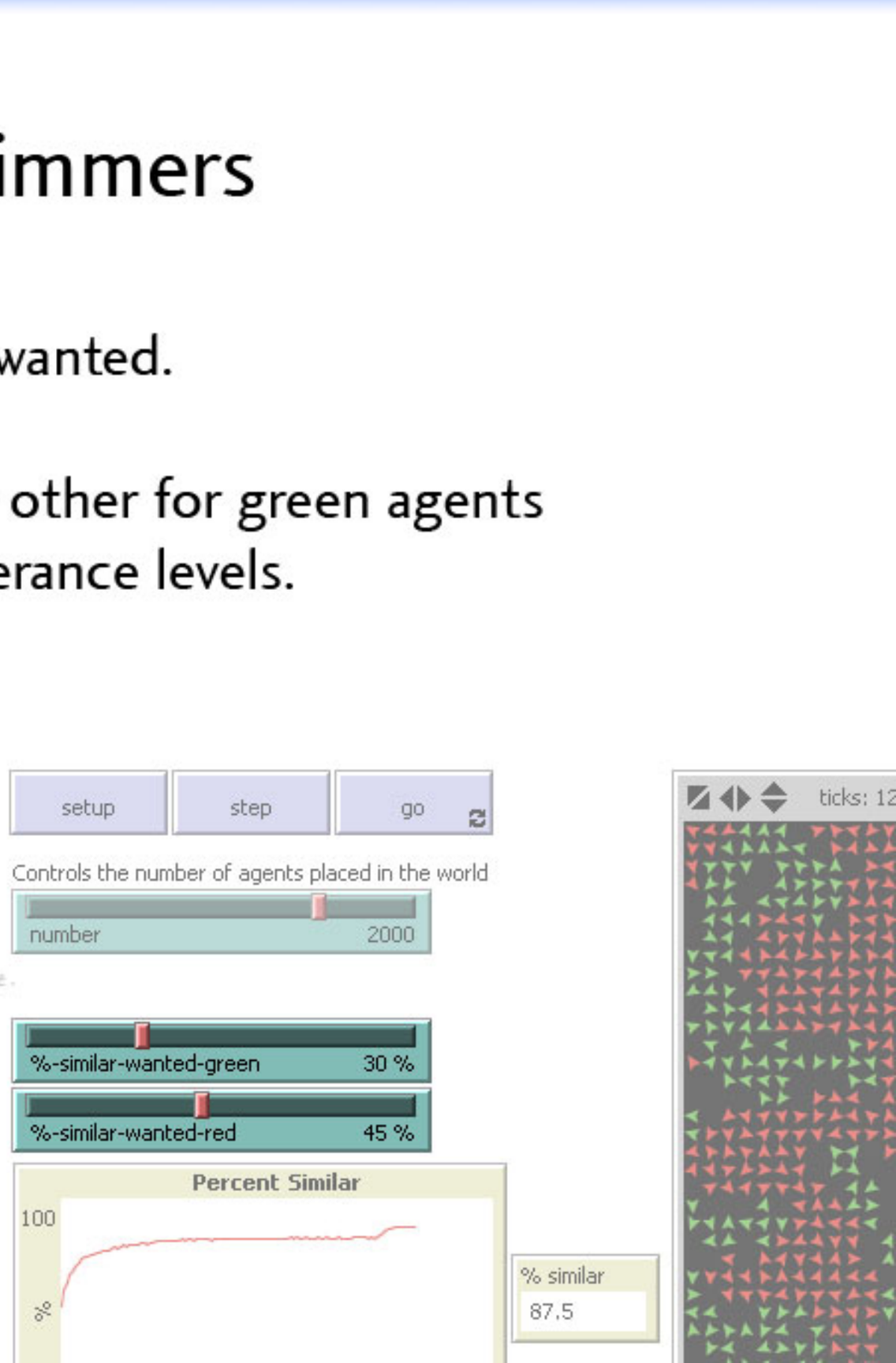
Add another slider for %-similar-wanted.

Make one for red agents and the other for green agents
so they can have different tolerance levels.

Extension 1: Surfers and Swimmers

- Add another slider for %-similar-wanted.
- Make one for red agents and the other for green agents so they can have different tolerance levels.

```
to update-turtles
  ask turtles [
    ;; in next two lines, we use "neighbors" to test the eight patches
    ;; surrounding the current patch
    set similar-nearby count (turtles-on neighbors)
      with [color = [color] of myself]
    set other-nearby count (turtles-on neighbors)
      with [color != [color] of myself]
    set total-nearby similar-nearby + other-nearby
    ;; the variable "happy?" is a boolean variable that has a value of either true or false.
    ;; if the condition after it is true then the value is set to true.
    ifelse color = green
      [ set happy? similar-nearby >= ( %-similar-wanted-green * total-nearby / 100 )]
      [ set happy? similar-nearby >= ( %-similar-wanted-red * total-nearby / 100 )]
  ]
end

;; these commands update the metrics collected and displayed on the interface.
;; they could actually be calculated in the monitors on the interface, but
;; calculating them here and just posting them there improves transparency.
to update-globals
  let similar-neighbors sum [similar-nearby] of turtles
```

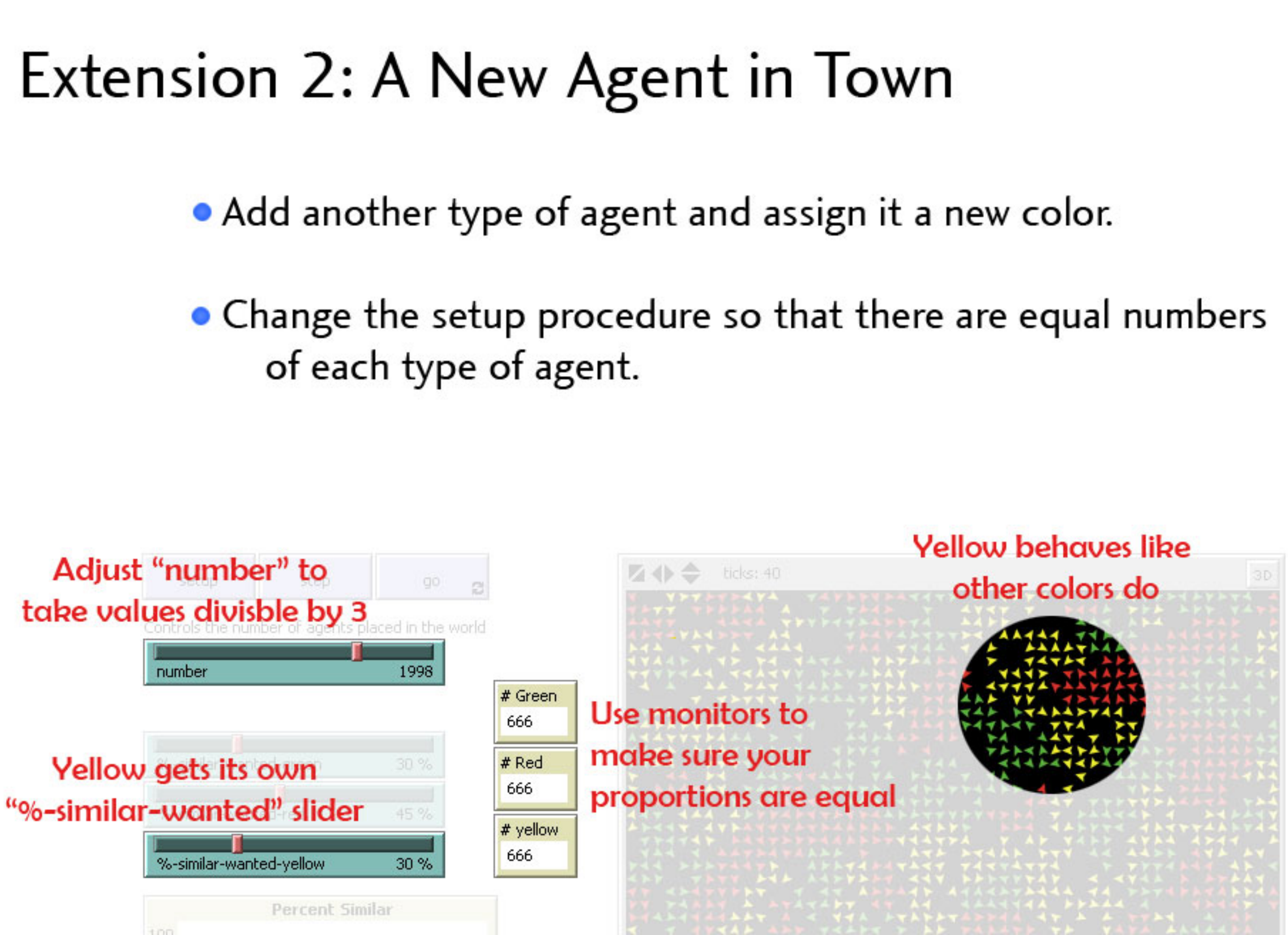


Extension 2: A New Agent in Town

- Add another type of agent and assign it a new color.
- Change the setup procedure so that there are equal numbers of each type of agent.

Extension 2: A New Agent in Town

- Add another type of agent and assign it a new color.
- Change the setup procedure so that there are equal numbers of each type of agent.



Extension 2: A New Agent in Town

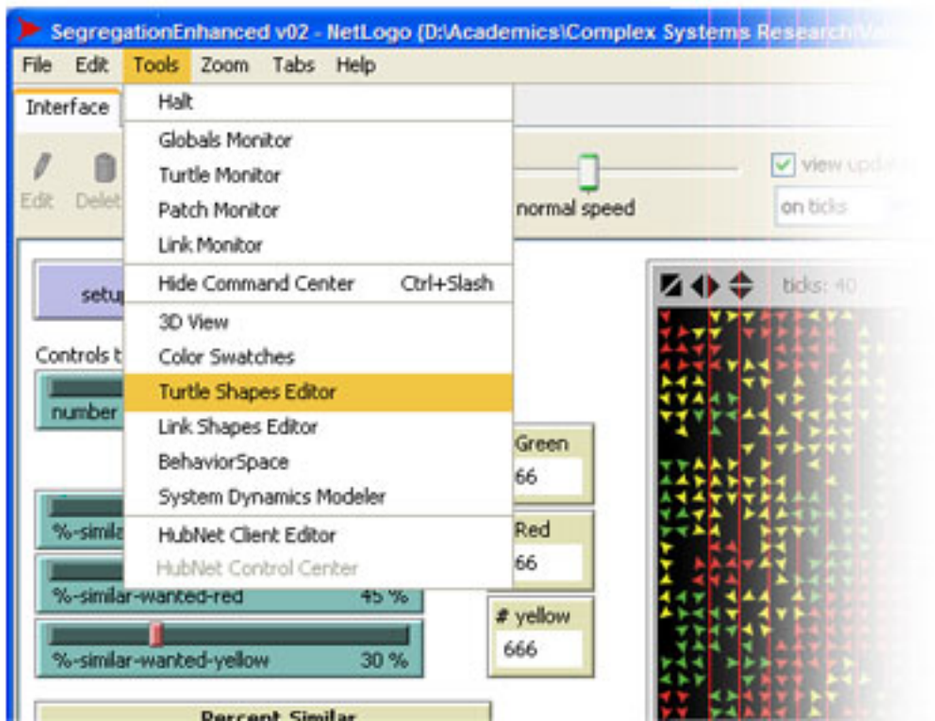
```
to setup
  clear-all        ;; removes all the turtles, clears plots, clears patch values, everything

  ;; this makes sure that you didn't specify too many turtles for the world-size selected...only one turtle per patch.
  if number > count patches
    [ user-message (word "This pond only has room for " count patches " turtles.")
      stop ]

  ;; create turtles on random patches...the patches are creating the turtles at their location.
  ask n-of number patches
    [ sprout 1
      [ set color red ] ]

  ;; turn a randomly selected third of the turtles green.
  ask n-of (number / 3) turtles
    [ set color green ]

  ;; now there are 2/3 red and 1/3 green; to make equal numbers convert only red agents
  ask n-of (number / 3) turtles with [color = red]
    [ set color yellow ]

  update-variables
  do-plots
end
```

```
to update-turtles
  ask turtles [
    ;; in next two lines, we use "neighbors" to test the eight patches
    ;; surrounding the current patch
    set similar-nearby count (turtles-on neighbors)
      with [color = [color] of myself]
    set other-nearby count (turtles-on neighbors)
      with [color != [color] of myself]
    set total-nearby similar-nearby + other-nearby
    ;; the variable "happy?" is a boolean variable that has a value of either true or false
    ;; if the condition after it is true then the value is set to true.
    if color = green
      [ set happy? similar-nearby >= ( %-similar-wanted-green * total-nearby / 100 )]
    if color = red
      [ set happy? similar-nearby >= ( %-similar-wanted-red * total-nearby / 100 )]
    if color = yellow
      [ set happy? similar-nearby >= ( %-similar-wanted-yellow * total-nearby / 100 )]
  ]
end
```

Extension 3: This Agent is a Transformer

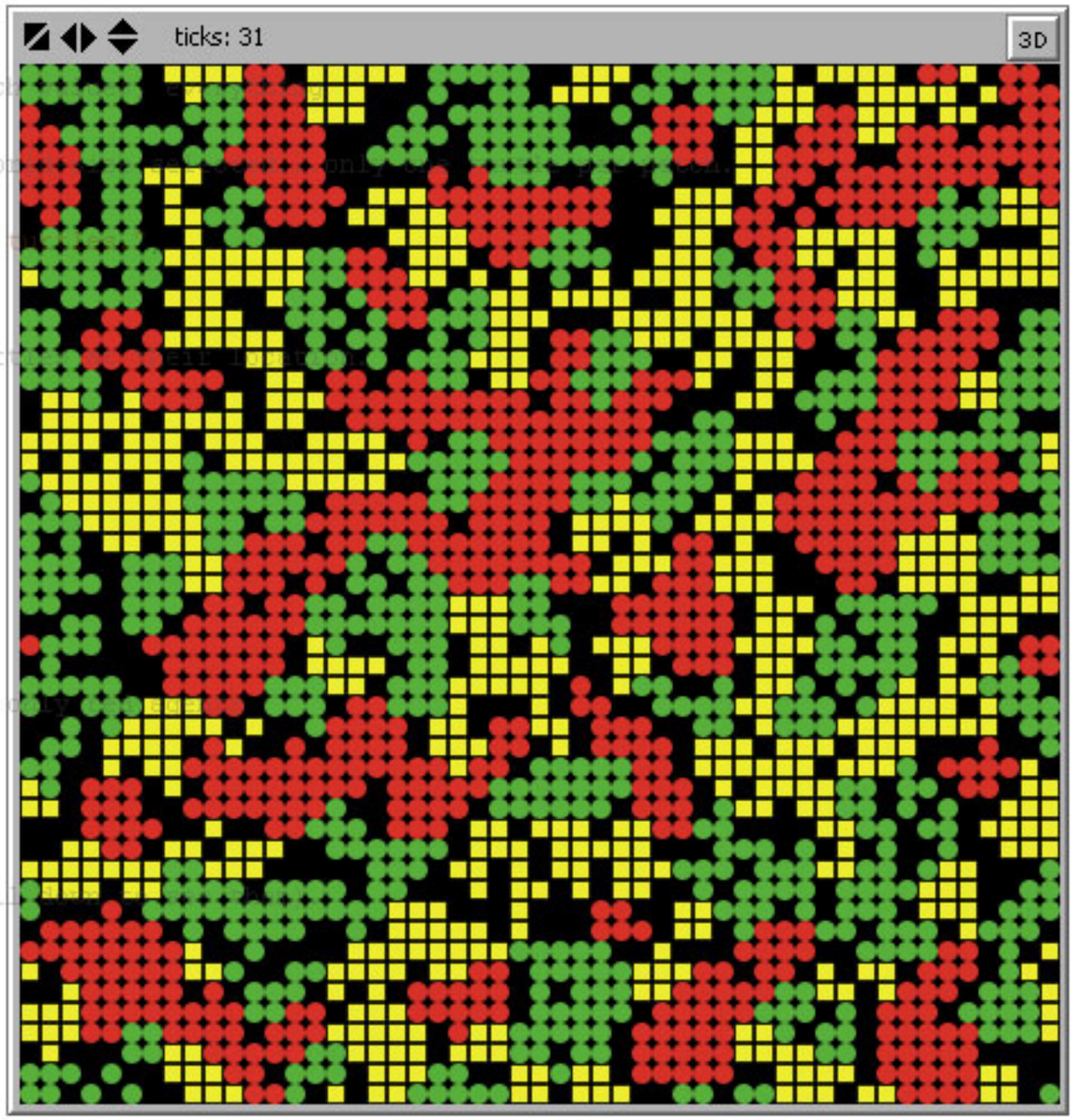
- Change the agents' shape so that it's easier to see the distribution of agents.

- Go to the tools menu and select the Shapes Editor and then choose the shape that makes sense to you.



Extension 3: This Agent is a Transformer

```
to setup
  clear-all        ;; removes all the turtles, clears plots, clears patch
  set-default-shape turtles "circle"
  ;; this makes sure that you didn't specify too many turtles for the wo
  if number > count patches
    [ user-message (word "This pond only has room for " count patches " t
      stop ]

  ;; create turtles on random patches...the patches are creating the tur
  ask n-of number patches
    [ sprout 1
      [ set color red ] ]

  ;; turn a randomly selected third of the turtles green.
  ask n-of (number / 3) turtles
    [ set color green ]

  ;; now there are 2/3 red and 1/3 green; to make equal numbers convert o
  ask n-of (number / 3) turtles with [color = red]
    [ set color yellow
      set shape "square" ]

  ;; these commands call procedures written elsewhere in the code (scroll
  update-variables
  do-plots
end
```

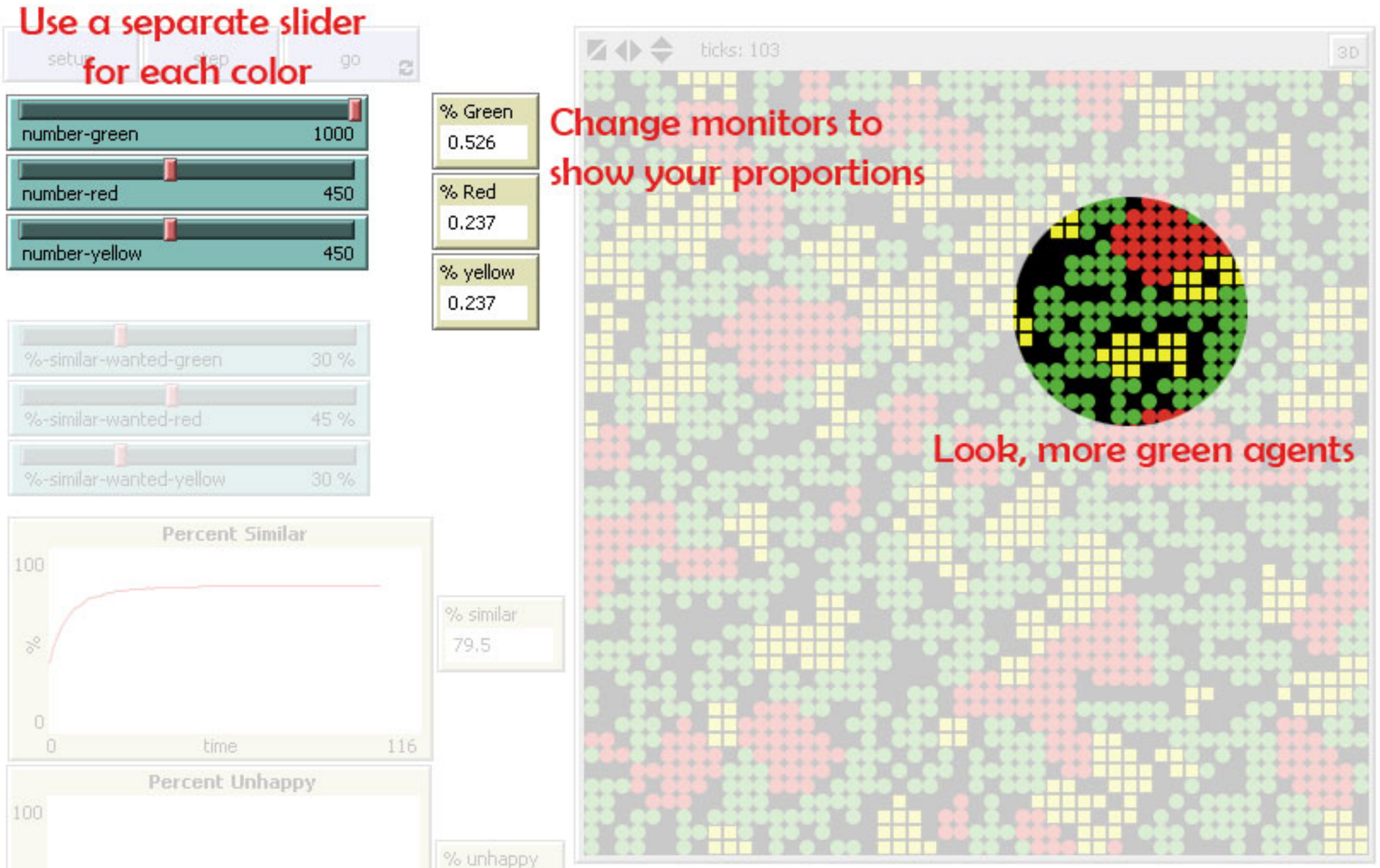


Extension 4: Not Created Equal (in Number)

- Add sliders for the number of each type of agent
  so you can have an uneven distribution of agent types.

- Run some experiments on different mixes and
  note any qualitative changes that result.

Extension 4: Not Created Equal (in Number)



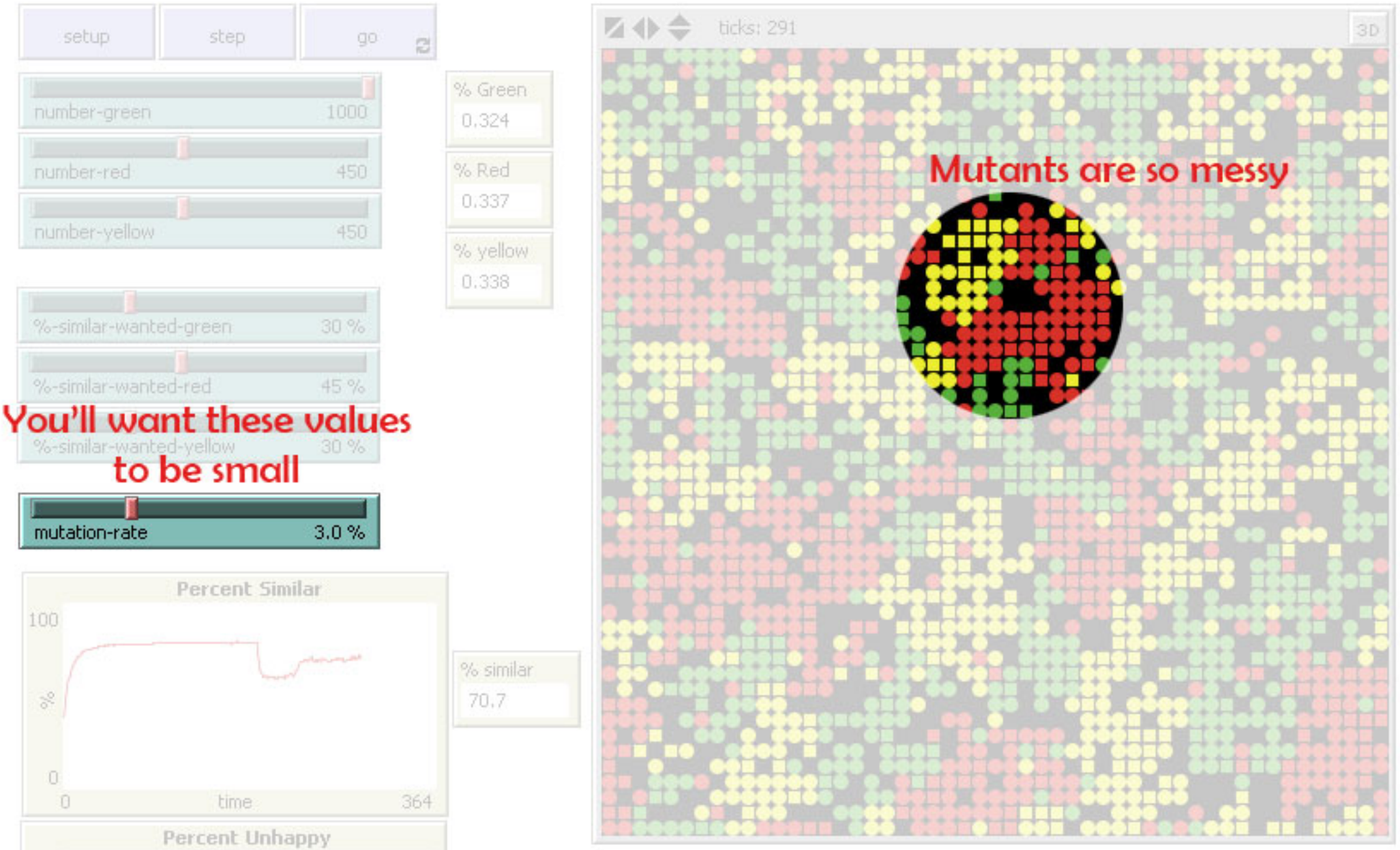
Extension 4: Not Created Equal (in Number)

```
to setup
  clear-all        ;; removes all the turtles, clears plots, clears patch values, everything
  set-default-shape turtles "circle"
  set total-number (number-green + number-red + number-yellow)
  ;; this makes sure that you didn't specify too many turtles for the world-size selected...only one turtle per patch.
  if total-number > count patches
    [ user-message (word "This pond only has room for " count patches " turtles.")
      stop ]

  ;; create turtles on random patches...the patches are creating the turtles at their location.
  ask n-of number-red patches
    [ sprout 1
      [ set color red ] ]

  ;; turn a randomly selected third of the turtles green.
  ask n-of number-green patches
    [ sprout 1
      [ set color green
        find-new-spot ]
    ]

  ;; now there are 2/3 red and 1/3 green; to make equal numbers convert only red agents
  ask n-of number-yellow patches
    [ sprout 1
      [ set color yellow
        set shape "square"
        find-new-spot ]
    ]

  ;; these commands call procedures written elsewhere in the code (scroll down to see them).
  update-variables
  do-plots
end
```

Extension 5: Beware of mutant agents

- Add a mutation rate so that unhappy agents have a percentage of changing their type rather than moving.
- Note that changing thier type may not make them happy depending on the percentage required.
- Does this speed up or slow down the time to universal happiness?

Extension 5: Beware of mutant agents



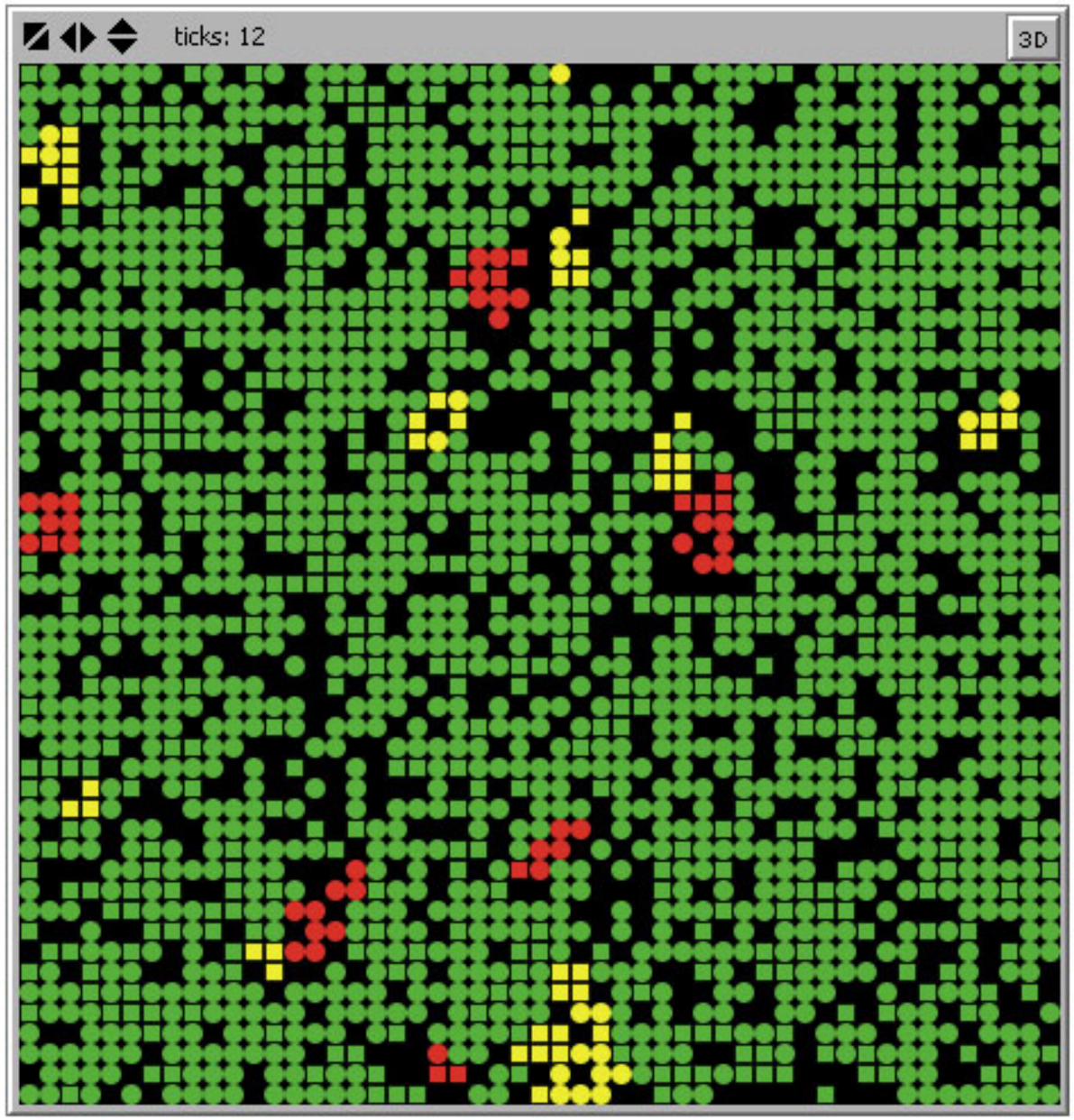
Extension 5: Beware of mutant agents

```
;; this method is run through each tick that the model goes through.
to go
  if all? turtles [happy?] [ stop ]    ;; if all the turtles are happy then the model breaks out of the go loop.
  mutate-turtles
  move-unhappy-turtles                 ;; if NOT all the turtles are happy, then the OBSERVER calls this method.
  update-variables
  tick                                 ;; this advances the time counter by one.
  do-plots
end                                    ;; this is the end of the go method,
                                       ;; but if a forever button is used it repeats from the beginning.

to mutate-turtles
  ask n-of (total-number * mutation-rate / 100) turtles [
    let current-color color
    if current-color = red [ set color green ]
    if current-color = green [ set color yellow ]
    if current-color = yellow [ set color red ]
  ]
end
```

Extension 6: Mutants can be students too

- Change the mutation rule into a learning rule so that agents copy the type of turtle that has the highest percentage of happiness among its neighbors.

Extension 6: Mutants can be students too



Extension 6: Mutants can be students too

```
;; this method is run through each tick that the model goes through.
to go
  if all? turtles [happy?] [ stop ]    ;; if all the turtles are happy then the model breaks out of the go loop.
  mutate-turtles
  learn-from-neighbors
  move-unhappy-turtles                 ;; if NOT all the turtles are happy, then the OBSERVER calls this method.
  update-variables
  tick                                 ;; this advances the time counter by one.
  do-plots
end                                    ;; this is the end of the go method.
                                       ;; but if a forever button is used it repeats from the beginning.

to learn-from-neighbors
  ask turtles [
    let happy-red-% 0
    if any? (turtles-on neighbors) with [color = red]
      [set happy-red-% count (turtles-on neighbors) with [color = red and happy?] / count (turtles-on neighbors) with [color = red] ]
    let happy-green-% 0
    if any? (turtles-on neighbors) with [color = green]
      [set happy-green-% count (turtles-on neighbors) with [color = green and happy?] / count (turtles-on neighbors) with [color = green] ]
    let happy-yellow-% 0
    if any? (turtles-on neighbors) with [color = yellow]
      [set happy-yellow-% count (turtles-on neighbors) with [color = yellow and happy?] / count (turtles-on neighbors) with [color = yellow] ]
    if happy-red-% > happy-green-% and happy-red-% > happy-yellow-%
      [set color red]
    if happy-green-% > happy-red-% and happy-green-% > happy-yellow-%
      [set color green]
    if happy-yellow-% > happy-green-% and happy-yellow-% > happy-red-%
      [set color yellow]
  ]
end
```

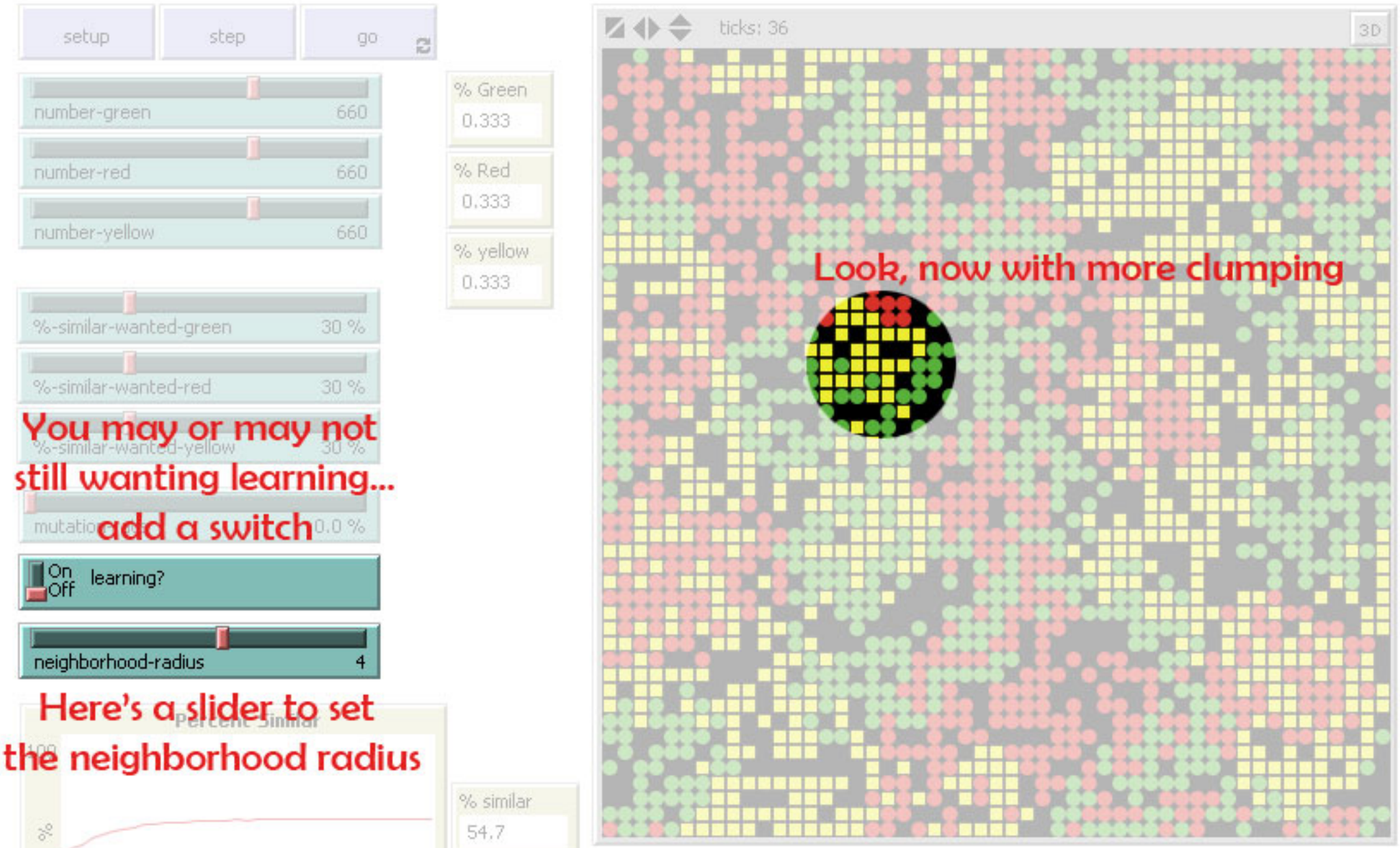
Extension 7: Not (just) in my backyard!

- Instead of just the Moore neighborhood,
  add a radius slider and have agents calculate happiness
  based on all the agents within that radius.

- Does this make it easier are harder to be happy?

- What effects does this have on the distribution of agents?

Extension 7: Not (just) in my backyard!



Extension 7: Not (just) in my backyard!

```
to go
  if all? turtles [happy?] [ stop ]    ;; if all the agents are happy then the model breaks out of the go loop.
  mutate-turtles                       ;; this activates the mutation method below
  if learning? [learn-from-neighbors]  ;; note that the order REALLY matters here...change the order of moving and learning to see what ha
  move-unhappy-turtles                 ;; if NOT all the turtles are happy, then the OBSERVER calls this method.
  update-variables
  tick                                 ;; this advances the time counter by one.
  do-plots
end                                    ;; this is the end of the go method,
                                       ;; but if a forever button is used it repeats from the beginning.
```

The switch variable holds a true or false value

Change “turtles-on neighbors” to this everywhere it occurs (hint: don’t forget the learning method)

```
to update-turtles
  ask turtles [
    ;; in next two lines, we use "neighbors" to [illegible]
    ;; surrounding the current patch
    set similar-nearby count (turtles in-radius neighborhood-radius) with [color = [color] of myself]
    set other-nearby count (turtles in-radius neighborhood-radius) with [color != [color] of myself]
    set total-nearby similar-nearby + other-nearby
    ;; the variable "happy?" is a boolean variable that has a value of either true or false.
    ;; if the condition after it is true then the value is set to true.
    if color = green
      [ set happy? similar-nearby >= ( %-similar-wanted-green * total-nearby / 100 )]
    if color = red
      [ set happy? similar-nearby >= ( %-similar-wanted-red * total-nearby / 100 )]
    if color = yellow
      [ set happy? similar-nearby >= ( %-similar-wanted-yellow * total-nearby / 100 )]
  ]
end
```

That's just the beginning!

Visit ComplexityBlog.com for more resourses on agent-based modeling including:

- Slides from my ICPSR ABM workshops
- Links to community sites and sources of help
- Blog posts on advanced techniques
- Coding Tips and Tricks to speed you along

And feel free to contact me for further advice, questions modeling help, collaborations, or fashion advice:

- aaronbramson@gmail.com

THANK YOU ALL