

ASSIGNMENT 4: FEEDBACKS AND PATH DEPENDENCE

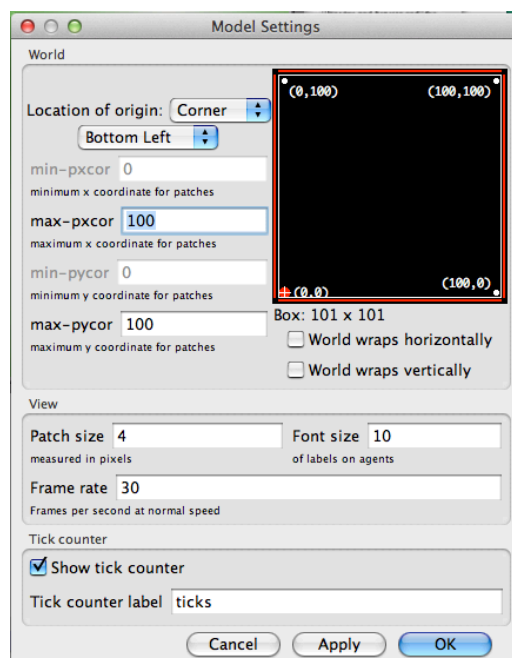
Objectives: (1) To learn excellent modeling technique by assembling a spatial model from its component sub-models. (2) To observe how feedbacks and path dependence shape the outcome of model runs.

Description: We will be building the model from Brown et al. (2005) **Path dependence and the validation of agent-based spatial models of land use**. This is a model of urban development that illustrates how both feedbacks and path dependence can affect the spatial structure of development over time. The model consists of several sub-models, which you will build and test individually.

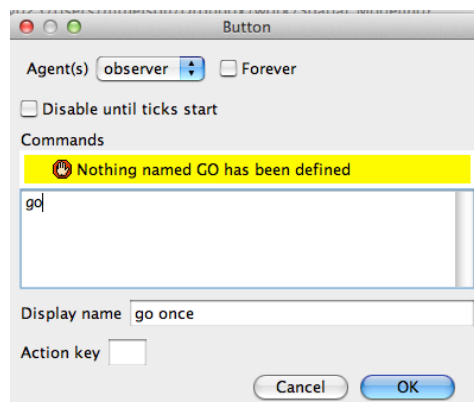
INSTRUCTIONS

PART 1: SETTING UP YOUR MODEL SKELETON:

1. Create new model.
2. For this model, we want a 100 by 100 grid of patches. Click on the “Settings” button on the top of the Interface tab in NetLogo and adjust the world settings to the following:



3. This model has a number of parameters that we will adjust among model runs once the code is complete. The function of each of these variables will become apparent as you fill out the model. Create the following sliders with minimum values of 0, maximum values of 1, and increments of 0.01:
 - a. `distance_services_preference`
 - b. `aesthetic_quality_preference`
 - c. `neighborhood_density_preference`
4. Create a slider, `ideal_density`, with minimum 0.5, maximum 1.0, and increment 0.01.
5. For some model runs, we will want to create an initial service center in the center of the world. Create the following switch:
 - a. `initial_center?`
6. Create buttons for the methods `setup`, `go`, and `go once`. Check the “Forever” box for `go`. The `go once` button setup window should look like this:



PART II: SETTING UP SUB-MODELS

7. The agents in this model will all be patches, so we need to set up the appropriate patch variables. A key aspect of developing a model is making sure your code is understandable, not only for yourself, but for others who will evaluate, use, and modify your model. The best practice is to make heavy use of comments in the code. Comments consist of strings of text incorporated into the code, which are ignored by the model interpreter. Although the NetLogo language was designed to be intuitive to read, it is still necessary to include comments in human-readable language for ease of communication. To make a comment in NetLogo, simply type a semicolon “;”. All text following the semicolon will be ignored by the

interpreter. I have included comments in some of the code, and I encourage you to add your own comments to the rest to help make your code more understandable. Use the following code to set up our initial patch variables paying attention to the descriptions of each variable in the comments:

```
patches-own
[
  status          ;; Current patch status:
                  ;; (empty, home, attraction,
                  ;; or service center).
]
```

8. You will need the following code to set up some global variables:

```
globals[
  max_dist      ;; Maximum possible distance between two
                ;; patches in the world
  aes_min       ;; Minimum possible aesthetic value
  aes_max       ;; Maximum possible aesthetic value
  most_recent   ;; Newest development
  n_centers     ;; Counter to keep track of service centers
]
```

9. You will recall from the Brown et al. paper that the agents in the model take into account three variables when choosing which empty locations to develop: **aesthetic quality** of the site, **distance to service centers**, and preferred **neighbor density**. Let's start with a model of aesthetic quality. We'll set up the model with two points of attraction located at the coordinates (25, 25) and (75, 75):

```
to setup
  clear-all
  reset-ticks
  ask patches[
    set status "empty"    ;; Initialize all patches
                          ;; to empty status.
  ]
  ask patch 25 25 [
    set status "attraction"
    set pcolor yellow
  ]
  ask patch 75 75 [
    set status "attraction"
    set pcolor yellow
  ]
end
```

In the Interface tab, click the setup button to verify that the yellow points of attraction appear where you expect them to in the world grid.

10. Next, we need to set the aesthetic quality for each patch, based on the proximity to a point of attraction. First add the following patch variable:

`aesthetic_quality` and add a comment with an appropriate description. Add the following lines to your setup method:

```
set max_dist (sqrt (2 * ( world-height ^ 2)))  
;; Maximum possible distance between  
;; two patches in the world.
```

11. The following method will calculate the aesthetic quality for each patch (be sure to add a call to `calculate_aesthetic_quality` to the end of your setup method):

```
to calculate_aesthetic_quality  
  ask patches [  
    set aesthetic_quality max list  
      0  
      (max_dist - distance min-one-of patches  
        with [status = "attraction"]  
        [distance myself]) / max_dist  
  ]  
  set aes_min min [aesthetic_quality] of patches  
  set aes_max max [aesthetic_quality] of patches  
end
```

12. Visualization of model parameters is an important way to make sure your code behaves as expected. One of the strengths of NetLogo is that it is relatively straightforward to visualize model parameters. Each patch has a value for its aesthetic quality, and we will use the built in function `scale-color` to visualize the relative aesthetic qualities of all the empty patches (lighter colors correspond to higher aesthetic values). Create a button called `show_aesthetic_quality`, and paste the corresponding method into your model code.

```
to show_aesthetic_quality  
  ask patches [  
    if status = "empty" [  
      set pcolor scale-color green  
        aesthetic_quality aes_min aes_max  
    ]  
  ]  
]
```

end

Experiment with the `show_aesthetic_quality` method and make sure you understand what is being displayed in the world grid.

13. Now that we have created our submodel for calculating and visualizing the aesthetic quality, let's create methods to calculate and display the distance from service centers. Make sure that the switch `initial_center?` is set to "On". Next we will add code to our `setup` method to create an initial service center:

```
ask patches [set distance_to_service 1]

if initial_center? = true [
  ask patch 50 50 [
    set status "center"
    set pcolor green
  ]
  ask patches [calculate_service_distance]
  set n_centers 1
]
```

14. Create a new patch variable `distance_to_service`. Create a button `show_distance_to_service`. Now to calculate and visualize the distance to the nearest service center for each patch:

```
to calculate_service_distance
  if n_centers > 0 [
    set distance_to_service
      1 - (distance (min-one-of patches with
        [status = "center"] [distance myself]) /
max_dist)
  ]
end

to show_distance_to_service
  let service_min min [distance_to_service] of patches
  ifelse count patches with [status = "center"] > 0
  [
    ask patches [calculate_service_distance]
    ask patches[
      if status = "empty" [
        set pcolor scale-color green distance_to_service
service_min 1
      ]
    ]
  ]
  [
  ]
end
```

```

ask patches [
  if status = "empty" [ set pcolor green]
]
end

```

Add a call to the method `calculate_service_distance` to your `setup` method;

```
ask patches [calculate_service_distance]
```

Experiment with visualizing the distance to service for the patches. Is this behavior what you expected? How is the pattern different from that of the aesthetic value? What happens if you turn `initial_center` to off and rerun `setup`?

15. We need to combine information about the distance to services as well as the aesthetic quality to calculate the attractiveness of empty patches for potential development. We'll formulate a utility function that uses the parameters `distance_services_preference` and `aesthetic_quality_preference` to weight the importance of both factors. We will add neighborhood density to this calculation after we introduce development. We will create a utility function based on equation 1 from the Brown et al. paper:

$$u_{xy} = q_{xy}^{\alpha_q} \times sd_{xy}^{\alpha_{sd}} \times (1 - |\beta_{nd} - nd_{xy}|)^{\alpha_{nd}}$$

where $q_{xy}^{\alpha_q}$ is the aesthetic quality weighted by the parameter `aesthetic_quality_preference`, $sd_{xy}^{\alpha_{sd}}$ is the preference for proximity to service centers weighted by `distance_services_preference`, and $(1 - |\beta_{nd} - nd_{xy}|)^{\alpha_{nd}}$ is the neighborhood density weighted by `neighborhood_density_preference`.

16. First add a patch variable, `utility`, and add a button `show_utility`. Our initial utility methods will be:

```

to calculate_utility
  calculate_service_distance

```

```

    set utility (aesthetic_quality ^
aesthetic_quality_preference) * (distance_to_service ^
distance_services_preference)
end

to show_utility
  ask patches [calculate_utility]
  let utility_min min [utility] of patches
  let utility_max max [utility] of patches
  ask patches[
    if status = "empty"[
      ifelse utility_min = utility_max
        [set pcolor blue]
        [set pcolor scale-color blue utility utility_min
utility_max]
    ]
  ]
end

```

Move the sliders for the service distance and aesthetic quality preferences to 0.5. Make sure that `initial_center` is switched back to “On”. Compare the patterns of aesthetic quality, distance to services, and utility. How are the three related? Explain what happens to the utility when you change the relative weights of distance to services and aesthetic quality.

PART 3: BEGIN DEVELOPMENT

17. Now it's time to start adding development to the model. Instead of allowing development on all possible empty cells, we want to limit the possible new development sites to a subset of the world's patches. Create a new input box for the variable `n_test` in the interface tab. Be sure to set the type to “number”. Set the value to 16. This quantity represents the number of randomly chosen vacant cells presented to a developer at each time step.
18. To add new development we need a method, `new_resident`, that will develop available cells. This method adds a housing development to the available patch with the highest utility.

```

to new_resident
  let potential_cells n-of n_test patches with [status =
"empty"]
  ask potential_cells [
    calculate_utility
  ]
  ask max-one-of potential_cells [utility] [
    set pcolor 125
    set status "home"
  ]
end

```

```

]
set most_recent max-one-of potential_cells [utility]
end

```

19. Create a `go` method to start adding houses:

```

to go
  new_resident
  tick
end

```

After running the `setup` method, add several houses using the `go` once button. Use the `show_utility` method to try to guess where new houses will be built. Try this with several settings of `distance_services_preference` and `aesthetic_quality_preference` to get a feel for how new house locations are chosen. What happens when you increase the value of `n_test` to 200?

20. Now that we are adding new developments, our utility function must take neighborhood density preferences into account. First add a new patch variable, `neighborhood_density`, and a method to calculate it:

```

to calculate_neighborhood_density
  set neighborhood_density 0.5 + 0.5 * (count neighbors
with [status = "home"] / count neighbors)
end

```

You must add this to the beginning of `show_utility`:

```

ask patches [calculate_neighborhood_density]

```

21. Our utility function must be updated to take into account neighborhood density. Change `calculate_utility` to:

```

to calculate_utility
  calculate_neighborhood_density
  calculate_service_distance
  set utility (aesthetic_quality ^
aesthetic_quality_preference) * (distance_to_service ^
distance_services_preference) * (1 - (abs (ideal_density
- neighborhood_density))) ^
neighborhood_density_preference
end

```

Run `setup`, add several houses, and observe how the spatial pattern of utility changes with different settings of `neighborhood_density_preference` and `ideal_density`.

- 22.** As new homes are added, we want to add new service centers. For every 20 new houses built, we will add a new service center at the closest open space to the most recently added house. Add the following method, and update the `go` method:

```
to add_service
  ;; start searching for empty spaces near the most
  recently added house
  ask most_recent[
    let radius_temp 1
    let check_neighborhood patches in-radius radius_temp
    while [count check_neighborhood with [status =
"empty"] = 0]
    [
      set radius_temp radius_temp + 1
      set check_neighborhood patches in-radius
radius_temp
    ]
    ask one-of check_neighborhood with [status = "empty"]
  [
    set status "center"
    set pcolor green
  ]
  set n_centers n_centers + 1
end

to go
  new_resident
  tick
  if ticks mod 20 = 0[
    add_service
  ]
end
```

To observe how this method works, run `setup` and then click `go_once` 19 times. On the next click, `go_once` will add a house and a new service center. Take a mental note of where both of these new developments occur. How might this spatial pattern affect future development?

- 23.** Finally let's add a counter for the number of houses and an upper limit for number of developments. Create a monitor box with the reporter set to `t i c k s`

and the display name set to **N Homes**. Create an input box named **max_homes**, set the type to “**number**” and set the initial value to 300. Add the following lines to the end of the **go** method to make your simulation stop after **max_homes** have been built:

```
if ticks >= max_homes [stop]
```

24. Now your model is complete, and you are ready to run experiments with different parameter settings to answer the questions in the next part!

PART 6: DISCUSSION

Create an Assignment 4 page. Provide videos, images (use File – Export – Export Interface...), and text as appropriate to address the following questions:

- i. Why is it important to test each of your sub models independently? Explain how you tested the sub-components of the main model using images and text to illustrate your methods and explain how your diagnostics conformed (or not) to your expectations.
- ii. The methods for calculating the components of utility are deterministic in this model. How is stochasticity implemented in the model? What parameter influences the degree of stochasticity?
- iii. How is initial environmental heterogeneity implemented in this model and what parameter or parameters determine its importance? Do you think that greater environmental heterogeneity corresponds to greater variability among replicate model runs, and why? Support your argument with evidence from replicate simulations with and without initial environmental heterogeneity.
- iv. Explain the roles of feedbacks and path dependence in this model. How are the two concepts related? How do the three parameters for preference relate to feedbacks and path dependence? What parameter settings would you use to eliminate any effect of feedbacks or path dependence?
- v. Describe at least two of the model assumptions or simplifications and how they could influence your interpretation of model results. Despite these assumptions and limitations, what can we learn from this model?

GRADING

Your answer to each question in Part 6

5 POINTS

TOTAL

25 POINTS

DUE DATE:

Tuesday, May 12th at 11:59pm

*Late submissions will be penalized 5% per day.